Vol. 4, No. 4 July/August 1986 $4.00

programmer's journal

The Resource Journal for IBM PC Programmers

WRITING MEMORY RESIDENT PROGRAMS IN C
by
William Hunt

OPTIMIZING ASSEMBLER FOR SPEED
by
Michael Abrash

Teach an Old Compiler a new trick

by Tom Swan

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Get a Grip on Assembly Language.

The Award Winning Visible Computer: 8088.

Assembly language programming is far from easy, but you don't have to be a genius to learn it. Let your PC teach it to you.

The Visible Computer: 8088 is an award-winning mix of book and software for mastering the elusive skills of assembly language.

It's an animated simulation of the PC's microprocessor that takes you inside the 8088 as it executes programs. You'll see how instructions are executed, not just the result.

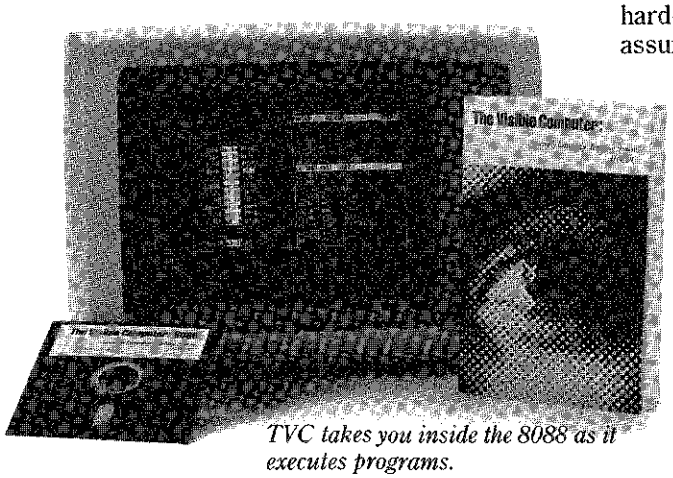
With its intuitive "moving bar" instruction display, memory windows, bulletproof beginner's mode, and more than 30 commands and functions, The Visible Computer is a powerful instrument for inspecting your own programs (or BIOS routines or anything else you can get into memory). You'll be using it as a debugging tool for years to come.

It's a book. The 350 page manual is more than simply instructions on working the simulator – it's a hard-working, assume-nothing tutorial on the fundamental concepts of assembly language. A lot of people think it's the best book on assembly language ever written.

PC Tech Journal "Program of the Month"

"(The Visible Computer) is at once useful and educational, offering an interactive debugger and a self-paced course in 8088 architecture and assembly language... A considerable value."

The Visible Computer: 8088 for IBM PC/XT/AT and true compatibles.



TVC takes you inside the 8088 as it executes programs.

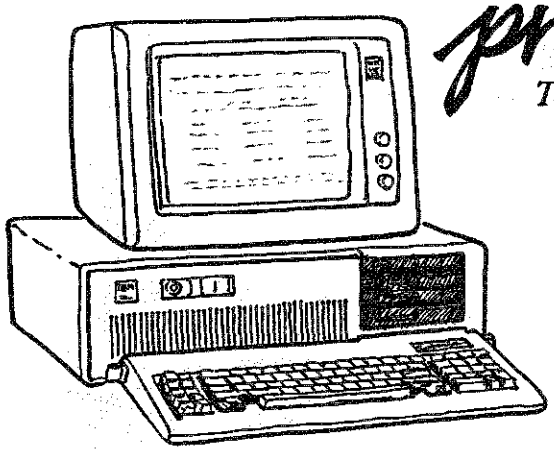
$49.95 Copy Protected

$79.95 Unprotected

If your dealer doesn't have it, order direct: Software Masters, 2714 Finfeather, Bryan, TX 77801 (409) 822-9490. Please enclose $3.00 shipping. Bank cards accepted.

Software Masters™

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



programmer's journal

The Resource Journal for IBM PC Programmers

Volume 4, Number 4 July/August 1986

CONTENTS

NOTE FROM THE EDITOR 2

LETTERS 4

PJ FORUM: DTACK GROUNDED, The Junk Mail Flyer 6
Adult toys, tomorrow's micros, HALGOL update, benchmarks, and RISCs

REDIRECTIONS: Domestic and Overseas Report by Frank D. Greco 14
Parallel computers, RISC, UNIX, and news from across the pond

CONSULTING: Efficient Custom Programming by Paul Barkley 18
Closer to a customizable program generator

DOS: DOS 2 File Handles and File I/O by Graham Pearson 22
A reference for file I/O function calls

UTILITIES: Programming the WordStar Document File Conversion Utility: Again and Again and Again . . . by Michael L. Hoyt 26
Convert WordStar to straight ASCII

ASM: More Optimizing for Speed by Michael Abrash 36
A follow-up of Michael Hoyt's article in the last issue of PJ

PASCAL: Teach an Old Compiler a New Trick by Tom Swan 40
Passing procedures as parameters

C: Memory Resident Programs in C by William Hunt 44
A guide to writing memory resident programs in C

BASIC: Tips on Formatting with BASIC PRINT USING Statements by Bernard H. Robinson, Jr. 50
A thorough survey of formatting BASIC output

dBASE: The Structure of dBASE II and III File Headers by Miriam Liskin 57
Access to the dBASE II and III file structures

MISC 62

PJ BILLBOARD 63

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

programmer's journal
The Resource Journal for IBM PC Programmers

Publisher: Elizabeth Oakley
Editor: Robert Keller
Managing Editor: Erin O'Connor
Production Manager: Ralph Bentley
Circulation/Promotion: Robert Tappan
Cover: Ralph Bentley
Illustrations: Julia King
Production Assistance: Holly Tracy

Contributing Writers:

Michael Abrash, Paul Barkley, Frank D. Greco, Hal W. Hardenbergh, Michael L. Hoyt, William Hunt, Miriam Liskin, Graham Pearson, Bernard H. Robinson, Jr., Tom Swan

Founder: Greg Estes

Business and Editorial Offices: Programmer's Journal, 29 West 29th Ave., Suite 5, Eugene, OR 97405, or P.O. Box 30160, Eugene, OR 97403, (503) 484-2162

Subscription Inquiries: Programmer's Journal, P.O. Box 3000, Dept. EE, Denville, NJ 07834, (503) 484-2162

Subscriptions: Bi-monthly rates $24 U.S. ($30 in Canada and $40 elsewhere) 2 years $44 U.S. ($55 in Canada and $75 elsewhere) 3 years $64 U.S. ($80 in Canada and $110 elsewhere)

Articles and letters in Programmer's Journal represent the views of their authors and not necessarily those of the publisher. Letters to the editor become the property of Programmer's Journal and may be edited and published.

Advertising: For ad rates, deadlines, and placement, contact Ralph Bentley or Robin Tappan, (503) 484-2162, or write to Programmer's Journal, P.O. Box 30160, Eugene, OR 97403.

IBM and Personal Computer are trademarks of International Business Machines; dBASE II and dBASE III are trademarks of Ashton-Tate.

Programmer's Journal (ISSN 0747-5861) is published bi-monthly by Oakley Publishing Company. Application to mail at second-class rates is pending at Eugene, OR.

POSTMASTER: Send address changes to Programmer's Journal, P.O. Box 3000, Dept EE., Denville, NJ 07834.

© 1986

NOTE FROM THE EDITOR

Mnematics Hosts a National *Programmer's Journal* SIG

We're now on-line with Mnematics, a nation-wide Videotex service. They can be reached in Sparkill, New York, at (800) 322-3633 and (914) 365-0184. Mention *PJ*, and for the proverbial limited time, joining is free.

You can reach Mnematics through Tymnet in over 600 U.S. cities for $4.50 per hour 6pm to 8am and $12.00 per hour 8am to 6pm. There is no surcharge for 1200 baud, and a 25% surcharge for 2400 baud, which is available in 65 cities.

Mnematics has been "quietly on line" since 1983, according to Gregg Squires, company spokesperson. The features of the service have expanded greatly over the last couple of months.

The first thing for all of us to do is determine just what structure the SIG should take. I have in mind a library of all the code from the magazine, with search and downloading by subject (language, issue, author, etc.); a library of reader contributions (we've received a number of articles that we can't publish because the code would take half an issue); a number of on-going forums; Peer Review voting; Reader Service response; a mailbox—what else? Let us know what you think.

We would also like to invest in a local bulletin board, and we want to make code available to the bulletin board operators who have inquired—Randy Gullick at (717) 226-6629 and others.

You can send us Email on Mnematics at 6601.

C Panel at the West Coast Computer Faire

Contributor William Hunt chaired the Saturday afternoon C panel in the Grande Ballroom at the West Coast Computer Faire in San Francisco. He covered these points in his introduction:

C is:

1. a fourteen-year-old general purpose programming language
2. a lot like BASIC, Fortran, PL/I, and Pascal
3. widely used and widely hyped
4. the most popular language for implementing software products for the PC
5. usable right now

A lot of what you read about C is superficial hype.

Why Use C?

1. Good compilers exist now
   -as opposed to compilers for new or fringe languages
2. You won't be alone
   -advice and help is available now from others working in the language
   -compilers are actively evolving and improving
   -there is a large market in related products—debuggers, librarians
   -the companies involved provide good support
3. C has practical features
   -separate compilation
   -modern data and control structures
   -low-level operations
4. C is open and understandable
   -no powerful but invisible support
   -the library is separate from the language
   -the library source code is available

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Reasons Not To Use C:

1. Software development is expensive—so use dBASE if possible.
2. A C hacker is a menace. C doesn't prevent you from doing dangerous things. It's for adults only. A hacker writing in C is like four teenagers in Dad's BMW.
3. C doesn't turn frogs into princes. It may be marginally better than other languages, but you still need knowledge.
4. If you are an ace with another language, e.g., Pascal, you should probably stick with that.
5. C is not tailored to specific applications. No ISAM, for example.

Usable C Compilers

1. Production Use:
   -Computer Innovations C86
   -Lattice C
   -Manx Aztec C
   -Microsoft C
   -Wizard C

2. Personal Use:
   -Datalight C
   -DeSmet C
   -Ecosoft C

In This Issue

Rex Jaeschke is taking a vacation from his ANSI C column. His column in the next issue will carry a report on the upcoming meeting of the ANSI C Standard committee.

In this issue, Michael Abrash further optimizes Michael Hoyt's screen dump code and includes a discussion of the 8088 pre-fetch queue.

—Robert □

COMBINE THE
RAW POWER OF FORTH
WITH THE CONVENIENCE
OF CONVENTIONAL LANGUAGES

HS/FORTH

Why HS/FORTH? Not for speed alone, although it is twice as fast as other full memory Forths, with near assembly language performance when optimized. Not even because it gives MANY more functions per byte than any other Forth. Not because you can run all DOS commands plus COM and EXE programs from within HS/FORTH. Not because you can single step, trace, decompile & dissassemble. Not for the complete syntax checking 8086/8087/80186 assembler & optimizer. Nor for the fast 9 digit software floating point or lightning 18 digit 8087 math pack. Not for the half megabyte LINEAR address space for quick access arrays. Not for complete music, sound effects & graphics support. Nor the efficient string functions. Not for unrivaled disk flexibility — including traditional Forth screens (sectored or in files) or free format files, all with full screen editors. Not even because I/O is as easy, but far more powerful, than even Basic. Just redirect the character input and/ or output stream anywhere — display, keyboard, printer or com port, file, or even a memory buffer. You could even transfer control of your entire computer to a terminal thousands of miles away with a simple >COM <COM pair. Even though a few of these reasons might be sufficient, the real reason is that we don't avoid the objections to Forth — WE ELIMINATE THEM!

Public domain products may be cheap; but your time isn't. Don't shortchange yourself. Use the best. Use it now!

HS/FORTH, complete system: $395. with "FORTH: A Text & Reference" by Kelly and Spies, Prentice-Hall and "The HS/FORTH Supplement" by Kelly and Callahan

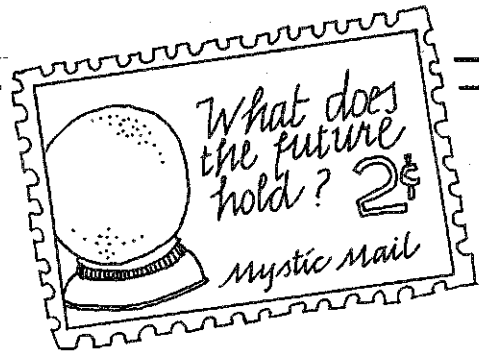
Visa Mastercard

HARVARD
SOFTWORKS
PO BOX 69
SPRINGBORO, OH 45066
(513) 748-0390

Circle Number 42 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LETTERS



Will PC/IX Languish?

I received my first issue of *Programmer's Journal* yesterday, and I just wanted to tell you that I really enjoyed Frank D. Greco's column "Redirections: Industry News." It was interesting reading and contained lots of useful information. One comment: I hope Frank Greco is right concerning IBM's releasing a System V version of PC/IX. I am a PC/IX owner, and I would like to see them upgrade the product from System III. It is a fine port of UNIX (in my opinion), and I have been very satisfied with it. I would hate to think that I spent quite a bit of money on a product, only to have IBM let the product die after a while.

I am looking forward to reading more of Frank Greco's column in the coming months.

Mark Davidson
Chattanooga, TN 37411

Funky Functions

In "Funky String Functions" by Tom Swan (*PJ*, March/April 1986), I was able (barely) to swallow passing strings as value parameters; at least Swan recognized that this slows things up considerably and rationalized his decision. But in the DeHead function, he makes one call to Delete for each blank to be removed, instead of merely counting the leading blanks and making a single call to Delete. And Swan actually says that it's all right to use a faulty function because range checking is normally off and the error will not be detected. I have news for Mr. Swan. Range checking is normally on while a program is being developed, and a function that will not work during program development is worse than useless as a utility function. Oh, I know I could turn range checking off locally around his function, but this is not the way to set up a utility library. It's not as if it would be a lot more difficult to write the function correctly, e.g.:

```
FUNCTION DeHead(s:OneString)  :OneString;
VAR
  n :  integer;
BEGIN
  n := 0;
  WHILE (n<Length(s)) AND (s[n+1] = Blank) DO n := n+1;
  if n > 0 THEN Delete(s,1,n);
  DeHead := s
END;
```

DeTail is poorly written, but at least not incorrect. The FoundChar variable is not needed:

```
WHILE (NewLen > 0) AND (s[NewLen] = Blank) DO NewLen := NewLen-1;
```

handles everything in a single statement.

Kenneth L. Lindsay
Laie, HI 96762 □

Tom Swan replies:

Yes, I played a trick—and documented it—with the string indexing. In my defense I'll say only that the functions work as I claim. Sometimes I break rules for advantage—when I can be certain of the consequences. If you prefer not to, don't.

Tom Swan
Lifitz, PA

So Much for So Little

I have just finished reading the excellent article by Mr. Robinson on BASIC's FIELD statements and felt compelled to comment. He touches on a facet of the popular microcomputer BASIC dialects that has been sadly overlooked—true screen and printer output control. It is unfortunate that such a state of affairs exists with this language.

IBM certainly didn't do the world any favors with its adoption of Microsoft BASIC. I'm sure that if they had had more communication within their halls, they could have easily ported their own MINI-BASIC to the PC family. Then dedicated people such as Mr. Robinson and countless others wouldn't have to stay up half the night just to output a few measly characters and numbers to the screen.

Peter Hoehler
New Hyde Park, NY 11040

Forethought Enough

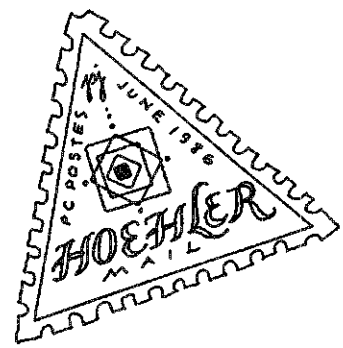
Thanks to Michael Shook for his good wishes upon the somewhat proximate occasion of Digital Acoustic's entry into the mass software market. ("Letters," *PJ* May/June 1986)

My DOS uses 1K sectors and 32-bit links to the next logical sector of a file. This means that a particular logical disk volume can address 4 times ten to the twelfth bytes. (Naturally, a physical disk can contain multiple logical disk volumes.) That's 4 terabytes or 4,000 gigabytes or 4 million megabytes. It is also 8,000 times as much data as one of the new 500 megabyte optical drives can hold. I am sure that this will some day prove to be inadequate and that my DOS will have to be re-written, but I do not think that day will come soon.

There *is* a DOS out there that badly needs fixing. . .while it is on its way to becoming UNIX, of course.

I confess to being the former editor of the late DTACK Grounded newsletter. But while most persons who write newsletters or columns are highly egotistical (after all, they assume that others will be interested in their writings!), I myself am very modest and unassuming.

Hal W. Hardenbergh
Chief B.S. Artist, Digital Acoustics, Inc.
Santa Clara, CA 92705



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Program faster, debug faster, in C, Pascal, BASIC, dBASE*

Now, use SOURCE PRINT to clarify and organize your source listings. You'll program more efficiently with this integrated source formatting tool:

Cross-reference Listing.

SOURCE PRINT's cross-reference list shows you relationships in your program by listing the line and page number of every occurrence of each variable, function and procedure. For each of these objects, SOURCE PRINT also lists all procedures in which the object is used.

Automatic Indentation.

SOURCE PRINT automatically indents source code based on structure nesting. This saves you time and keeps your source codes uniform. Alternatively, you can keep your source left-justified for speed and let SOURCE PRINT auto indent the listing.

Structure Outlining.

SOURCE PRINT automatically draws lines around nested structures on your source listing printouts, and on your monitor. Your program structures become clearer.

Nesting-error Detection.

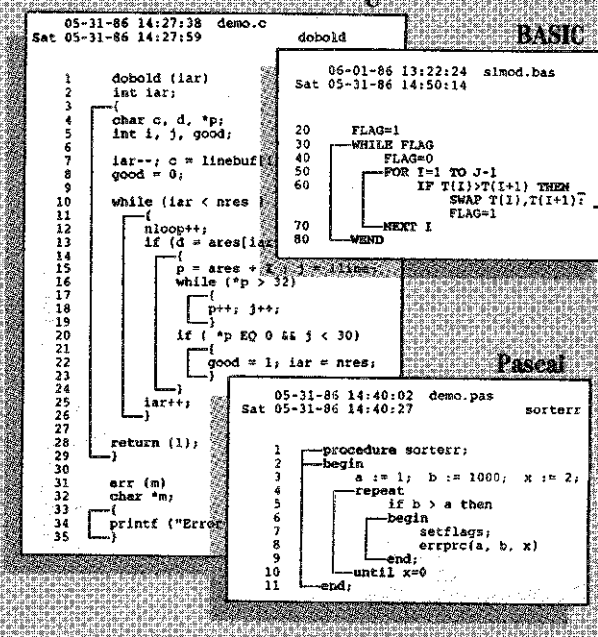
SOURCE PRINT flags structure nesting errors.

SOURCE PRINT also lets you print key words in boldface and display them emphasized or in color on your monitor, extract functions and procedures by name, generate page headings and a table of contents.

SOURCE PRINT is easy to use. All combinations of features are simply specified on the command line.

For IBM PC, XT, AT, compatibles. DOS 2 and later. ≥192K RAM. Not copy protected. Execute from floppy or hard drive.

*SOURCE PRINT works with dBASE II, III, III+ and Modula-2.



Order by mail or phone

800-257-5774 (within CA) **800-257-5773** (outside CA)

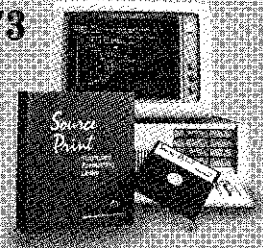
or see your local dealer

Call for free brochure

Aldebaran™ Laboratories Inc.

Developers of high-speed professional software

8738 Mt. Diablo Blvd., Lafayette, CA 94549 (415) 283-7084



Source Print™

Comprehensive Formatting Tool

$139.00 Version 1.2

License for use on a single computer.

Add $5.00 shipping/handling. Within CA add sales tax.

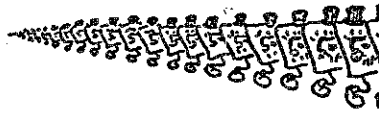
MC, Visa, AmEx accepted. Immediate delivery.

SOURCE PRINT is a trademark of Aldebaran Labs, Inc. dBASE is a trademark of Ashton-Tate

PRODUCED 2004 BY UNB ELECTRONICS LIMITED

Circle Number 52 on the Reader Service Card.

DTACK GROUNDED, The Junk Mail Flyer



Adult toys, tomorrow's micros,
HALGOL update, benchmarks, and RISCs

Not copyrighted by Digital Acoustics Inc.

You are reading junk mail again . . . congratulations!

One characteristic of junk mail is that you don't pay for it. Another is that you can't get off the mailing list, even if you want to. Yet another is that you can't get ON the mailing list, even if you want to!

But Seriously, Folks. . .

We call your attention to David Bunnell's column in January *MACWORLD*, in which he dumps on the Amiga. On page 16 David asks: "Why buy a machine with an entirely new operating system and no software base?" You will remember that Bunnell began publishing *MACWORLD* (using CWC's money) two years ago. The MAC was then an insanely great computer with an entirely new operating system and no software base.

Adult Toys:

A lot of software publishers (and book publishers) are becoming disenchanted with the Macintosh aftermarket and are looking elsewhere for sales. It seems, although it is generally agreed that the Mac has an installed base of 400,000 units, that Mac software and book sales are a heck of a lot lower than that installed base would suggest. The reason is very simple: Most Macs were bought as adult toys and are now gathering dust. You don't buy software or books for an adult toy that is gathering dust, just as I have not purchased an ocular or other accessory for my Celestron for a long time now.

Now, before some of you Mac owners puff up like horned toads, I know that *some* Macs are actually being *used*. The thing is not many of those 400,000 owners are like you, and so not much new Mac software gets sold.

A New Ball Game?

The new Mac Plus, at $2600, is almost as good a machine as the new Atari 1040 ST at $1000. The biggest difference is in the graphics: The monochrome ST has 32,000 pixels vs. under 22,000 for the Mac, the ST has graphics chip support and so draws lines up to 30 times faster than the Mac depending on the angle (slant) of the line, and the ST has a normal-size CRT instead of the teeniny "opthalmologist's friend" on the Mac. It will be interesting to see how many unreasoning Mac loyalists will be willing to pay 2.6 times as much money for an inferior machine.

Tomorrow's Micros Include:

Motorola 68020
Intel 80386
Nat Semi 32000 series
Zilog Z80000
AT&T 32100 chip set
Fairchild Clipper
INMOS Transputer
Acorn RISC (ARM)
AMD 32-bit-slice

As always, some of the new-generation 32-bit micros will prove to have better performance than others, and some will prove to be more successful in the marketplace than others. Here is my opinion of the contenders:

Zilog Z80000

The Zilog Z80000 has finally arrived in first silicon and is in "limited sampling." Zilog itself has officially announced that the Z80000 will not compete with the 68020 and the 80386, but will provide an upgrade path for the numerous imbedded Z8000s that got designed into military systems. Once again, I did not say that; *Zilog* said that!

Rumor has it that Zilog was given a year to make it—or else!—by parent EXXON, which has finally tired of Zilog's continual losses. More than a third of that year has already ticked away. . .

AT&T 32100 Chip Set:

The AT&T 32100 microprocessor is part of a very expensive 5-chip set. According to AT&T's ads, all five of those chips currently exist. (I'd bet a pretty penny that not all are in production!) The floating-point math chip does not perform transcendental calculations, like the Nat Semi chip and unlike the 68881 and the 8087. This chip set makes an excellent CPU for high-end minis that specialize in communications. As an efficient, cost-effective bit-banger or number-banger, *it is worse than the pits!*

Jack Tramiel is trying to negotiate a favorable contract with Motorola for delivery next year of lots of cheap 68020 and 68881 chips for a mass-market engineering workstation. Many of you may have seen recent articles announcing that Tramiel was negotiating with AT&T for their 32100 chip set. Baloney! Jack has to have *something* to negotiate down the 68020/68881 prices, and the Nat Semi chip set (32X32/32081) is no longer a credible alternative to the 68020/68881.

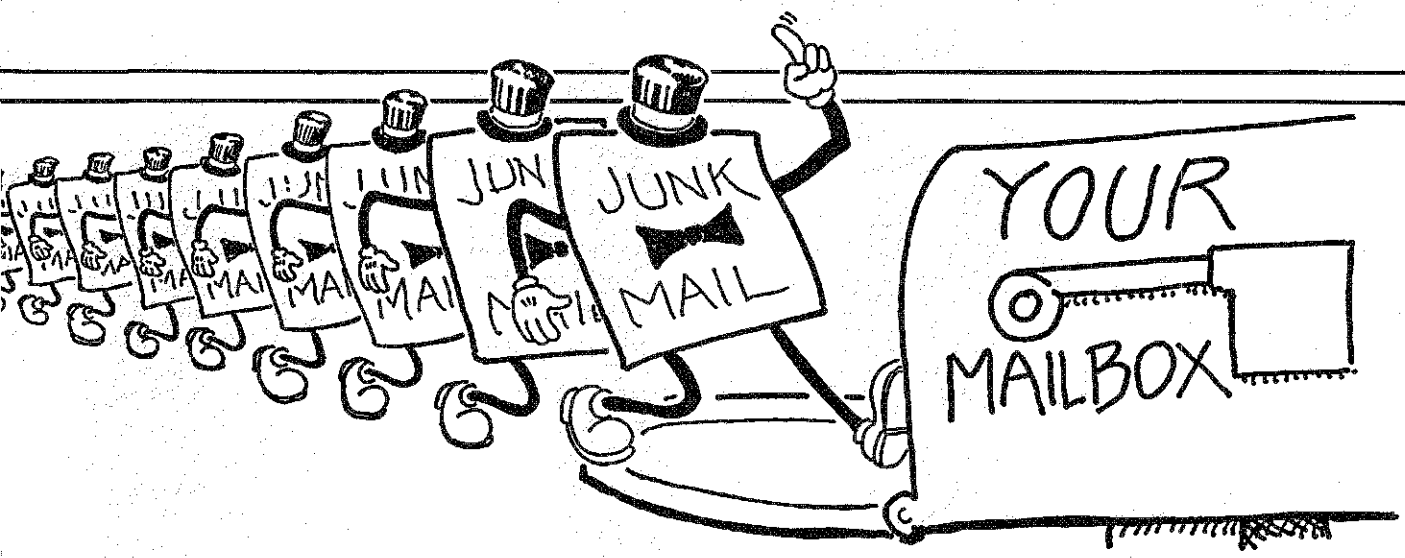
You will not ever see the AT&T 32100 in anything that pretends to be a personal computer.

Nat Semi 32000 series:

The 32332 is the latest wonderful from Nat Semi, which has *never* sourced a successful mass-market micro. The 32032 has already failed in the marketplace. The next Nat Semi part to make it in the micro-mass-market will be its *first*! (Remember that Nat Semi was selling its IMP-16 16-bit micro chip set *before* Mos Tech introduced the 8-bit 6502 that Chuck Peddle designed at Motorola.)

We told you before that everyone who is in a position to dump projects based on the 32000 series has done so, notably including Tektronix. The latest outfit to dump the 32000 series is its second source, T.I. A few months back T.I. was planning to duplicate the Nmos-based 32000 series in CMOS, was planning to introduce new, proprietary extensions to the chip set, and was vigorously

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



bidding on 32000 design-ins (including the six 32000-based graphics terminals Tektronix had announced?). T.I. has now dropped *all* of that, *including* the vigorous marketing effort. T.I. will now merely second-source those chips that they are *required* to second-source under the terms of a contract between T.I. and Nat Semi that was signed when J. Fred Bucy was running T.I. Since T.I.'s sales activities have dropped essentially to zero on the 32000 series, it is most unlikely that T.I. can be considered an *active* second source on the 32000 series.

When he was at the top of his form, the great sprinter Jesse Owens was asked whether he was confident of defeating a particular rival in an upcoming competition. Jesse replied, "Yes." Why? his questioner continued. "Because he has a defect in his running form," Jesse replied tactfully. What is the defect in his form? his questioner pursued. "He spends too much time running in the same place," Jesse explained.

The above paragraph just might give you a clue why people and companies are dumping the 32000 series right and left.

AMD 32-bit-slice:

We have listed AMD as making a bit-slice type 32-bit micro. Actually, AMD is merely the most visible of several chip-makers active in the bit-slice market. Bit-slice machines are microprogrammed, so you get to invent *your very own proprietary assembly language* for every new machine. These devices make superb disk controller and graphics controllers and work well in other specialized applications such as digital signal processing. Lots of moderately successful mid-performance minicomputers have been, and are, based on bit-slice technology. Every now and then somebody tries to make a personal computer using bit slices (Jerry Pournelle's treasured *LILITH* and the late HEX29), but none of these attempts has been successful in a mass-market sense.

(Besides AMD, both T.I. and Analog Devices are coming out with a complete 32-bit chip set. All three feature floating-point processor chips; the Analog Devices chip set features 64-bit (!) FP processors, and the 32-bit CPU has the primary purpose of calculating the address of the next 64-bit argument (during matrix inversions with pivoting, for instance). In other words, the Analog Devices chip set seems to be aimed at building very high-performance 64-bit FP number-crunchers *as attached processors* to conventional general purpose computers. AMD and T.I. are building very fast chip sets with 32-bit FP processors. None of these chip sets is complete yet. The T.I. FP chip will feature 1 micron design rules, and if you believe it will be finished this summer as announced, then I have reserved some choice Florida real estate just for you. . .Oh, yes: All of these FP chips are *fast*, which means none of them will directly execute an FP divide. And let us not forget that *many* dedicated applications do not require floating point support.)

Acorn RISC (ARM):

Acorn's (ugh!) ARM RISC microprocessor was inspired by the 6502. Unfortunately, the ARM's designers included too many 6502 "features," including the absence of a multiply or divide instruction. What that means is that the ARM will require *at least* 130 instructions to execute a 32-bit multiply; the number may be much larger. You can expect the ARM to execute lots more instructions in, say, one second than, say, the 68020. But the 68020 will most certainly get more useful work done in that same second.

The ARM is interesting because it is the first merchant microprocessor with built-in logic to take advantage of the page-mode capability of modern DRAMs and therby runs 30% faster than would otherwise be the case. There is going to be more of this appearing from other sources, soon. It doesn't take a genius to read one of Hitachi's two-page ads for its 256K DRAM (which features 50nsec read access in static CAS mode) to figure out how to most economically feed today's 20MHz 68020 or tomorrow's 25MHz part.

I believe that the 1985 ARM is very like 1979's Zilog 8000: a good but not great part that won't make it in the mass market.

INMOS Transputer:

A little background on the INMOS Transputer: INMOS was funded by the British government largely to develop the Transputer, which was going to revolutionize the computer industry (not just the mass-market microprocessor industry). Eventually, INMOS proved too expensive for the British government to keep (INMOS lost *lots* of money), and the British government put INMOS on the block. AT&T was an ardent suitor for INMOS, wanting INMOS' up-and-running wafer fabs, including the one in Glasgow, Scotland. However, the Brits didn't want to sell "another piece of the family silver" to the uncouth Yanks, so the British conglomerate Thorn/EMI bought INMOS instead.

And then the 64K DRAM market went to hell, INMOS shipped a great many defective static RAMs to its customers, all of the INMOS founders except Ian Barron were sacked, and Thorn/EMI wrote down the value of INMOS on its books to a negative value. (In other words, Thorn/EMI certified to the taxing authorities that INMOS was literally worth less than nothing!) Oh, yes: The chief

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

executive of Thorn/EMI at the time the decision was made to acquire ain't there no more. INMOS sole-survivor Ian Barron has publicly asserted that the future of INMOS is inextricably tied to the success of the Tranputer, which is now a real part and is being sampled. That's the background.

The principal Transputer chip is the 32-bit T-414 although 16-bit T-212s exist and future floating-point Transputers have been discussed. Proponents of the Transputer assert that (for instance) 100 Transputers can be interconnected and, using the computer language OCCAM, will result in 100 times higher performance than a single Transputer *for general purpose calculations.*

Persons such as myself, who are not proponents of the Transputer, believe that the Transputer can indeed provide very high performance for certain dedicated applications, using a particular dedicated wiring arrangement to interconnect multiple computing units. (See, for example, *BYTE*'s article on the Transputer, which showed as an example an obviously specialized physical interconnection for a dedicated digital signal processing application.)

It is a simple fact that nobody has yet come up with an architecture which is better for *general purpose* calculations than the ubiquitous von Newmann architecture. It turns out that the Transputer is just one of the many attempts in progress to successfully harness multiple microprocessors for computations. It ain't been done yet. (Miracles *can* happen, one supposes.)

A humorous aspect of the Transputer marketing effort is INMOS' refusal to provide assembly language documentation for the Transputer. (Anybody who wants to buy a merchant microprocessor for which the assembly code is a secret, raise your hand and I'll give you a dunce cap.) In fact, originally you were not permitted to program the chip using any language other than OCCAM although INMOS has backed down from that and now promises support for other traditional languages such as FORTRAN, C, and PASCAL. (Let's see: There was the Intel iAPX432, which you were not permitted to program using any language other than ADA . . .) If the Transputer is successful in any *general purpose* computing application, such as a mass-market personal microprocessor, it will surprise the hell out of me.

Fairchild Clipper:

Next we come to the Fairchild Clipper, which was originally billed as a RISC machine. Apparently Howard Sachs and his fellow Cray expatriates are not card-carrying RISC supporters. It turns out that, while certain instructions *do* execute in a single clock cycle (your fundamental RISC dialectic), the Clipper is rated by Fairchild at 5.5 mips, which means the *average* instruction requires 6 clock cycles. In other words, the Clipper may be more a CISC than a RISC. (While the Clipper was being developed, "RISC" was the hottest buzzword in computing, as "UNIX" was in late '82. Naturally, Fairchild preferred to apply this then-hottest buzzword to its new CPU development.)

This machine has a chip set that is sold as a board-level module. Don't shake your head—this is a *good* idea! Right now the Clipper is in "limited sampling," which means Digital Acoustics could not get its hands on one if it wanted to. But this is a very interesting part and appears at first glance to be superior in performance to both the 68020 and the 80386. That's at first glance. When the part becomes more readily available, I plan to take a closer look at it. Will it support double precision floating point or just single imprecision? Will it incorporate logic to take advantage of static CAS DRAMs? What languages will be available to support the module? With what price tags?

The real question is whether the Clipper is just a packaged 32-bit-slice machine like AMD's or T.I.'s. If so, that 30nsec clock period is damned impressive! I think the Clipper might be a sleeper, especially if Fairchild can get its price tag down. . . .

The Bobbsey Twins:

Yes, the 68020 and 80386. If either one of these devices did not exist, I could immediately assert that the other was unquestionably the best single-chip merchant microprocessor around. However, both (sigh) exist, and that makes life a lot more complicated.

I am personally very interested in performance; most of you know that. O.K., which of the Bobbsey Twins has higher performance? Well, that depends. What you do is adopt a set of assumptions and hold your head just so, and one chip turns out to be a few percent faster than the other. Hold your head differently, and you might get a different answer. What's that? You are looking for definite answers? Sorry, I'm fresh out.

Here are some of the differences between the twins:

The 68020 is in production today. You can buy a thousand 12.5MHz MC68020s and get prompt delivery, or you can buy a hundred 16MHz XC68020s and get reasonably prompt delivery. The 80386 has just begun limited sampling. The 68020 has at least a 14-month lead on the 80386. We say at least because Motorola worked the "bugs" out of the 68020 very quickly as new, complex chips go, but Intel has not yet worked the bugs out of its *last* generation 80286!

Let up hypothesize an advanced version of the 68020, which I will call the 68050. Let us suppose that the 68050 will apear 28 months behind the 68020—about 2½ years later. Then the comparison of the 68050 with today's 80386 would be as valid as comparing the 68020 with the 80386 since the 68050 would be only 14 months behind the 80386! And if the 68020 and the 80386 are very close in performance, the 68050 would unquestionably outperform the 80386. What I am saying is that that 14-month (at least) lead which the 68020 has is *important!* Tomorrow's microprocessors *should* logically outperform today's, no?

Both the 68020 and the 80386 will, when in full production, have 16MHz clocks (at least). Both CPUs are much too fast for use with ordinary DRAM mass-memory without being seriously slowed by that mass-memory. This means that both chips seriously need internal logic to work with static CAS mode DRAM. But neither chip has such logic yet.

A way to lessen the impact of slow mass-memory is the use of small, high-speed cache memory. The 68020 has a 256-byte cache built-in, and the 80386 does not. Advantage Motorola. If a larger external cache is installed with high-speed static RAM, the 80386 can access that cache in two clocks instead of three clocks for the 68020. Advantage Intel. (But that had better be a *very* fast external cache and ancillary logic.)

Since Intel can access external caches in two clocks vs. Motorola's three, Intel likes to point out that that means the 386 has a bus-bandwidth of 33.3 Mbytes/sec vs. only (!) 22.2 Mbytes/sec for the 020. Motorola replies that the 020 uses a "Harvard" internal architecture, meaning that there are two separate internal 32-bit buses, one for instructions and one for data. That means that the 020 can simultaneously access its internal cache at 33.3 Mbytes/sec for instructions while accessing an external cache at 22.2 Mbytes/sec for data. That makes a total bus-bandwidth of 55.5 Mbytes/sec!

The instruction set of the 68020 is a superset of the 68000's while the instruction set of the 80386 is a superset of the 8080's. Advantage Motorola. There is a lot more 8080 code around to be

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

a superset *of* than there is 68000 code. Advantage Intel.

All of the preceding comparisons were nice and simple and straightforward. Now it gets more complex. You see, it is becoming evident to everybody with an I.Q. one higher than a rock that a general purpose microprocessor really should be augmented by a floating point math chip capable of double precision. These math chips come in a number of flavors:

1. Is the chip a peripheral or a coprocessor? This is a "don't care" in a single-tasking environment but is *vitally* important in an interrupt-driven multi-tasking environment. More on this later.
2. Does the chip execute transcendentals directly, as does the 68881, some transcendentals directly as does the 8087, or no transcendentals, as (doesn't) the Nat Semi and AT&T math chips?.
3. Can the chip directly execute a floating point divide? (Don't laugh; Weitek, Analog Devices, AMD, and T.I. are all making FP math chips that cannot perform a divide.)
4. Does the chip exist as a bug-free production part? (8087, 32081) Does it exist as real silicon, but with bugs and not in production? (68881) Is it a pure fantasy, existing only as a part number? (80387).
5. Is the math chip slow by today's standards? (8087) Average by today's standards? (32081) Fast by today's standards? (68881 and the literally fantastic 80387) *Blindingly* fast? (Weitek, Analog Devices).

The fastest double-precision FP math chip that will perform a divide and that can be purchased in a bug-free form by the hundreds or thousands *now, today* (20 Jan '86) is the Nat Semi 32081. The 32081 works with the Nat Semi 32000 series as a coprocessor and with all other microprocessors as a peripheral.

More Resident 68K Assembler:

We reported that the QD DOS version of our resident 68000 assembler required 74 seconds to assemble HALGOL logical disk #1 as opposed to 463 seconds for the Phase Zero cross-assembler. By some fine-tuning, we have reduced the time for the QD DOS resident assembler to 53.8 seconds if the object code is to be saved to a binary file, or 45 seconds without the binary save. This was accomplished by adding BLOADZ and DATALDZ to QD DOS, where the added Z indicates that the disk catalog is not to be read, and by arranging the assembler program and the pass1 and pass2 object files so they each lie entirely within a single QD DOS 8K pseudofile.

We asserted that James and I were going to develop a real DOS based on the 1.28Mb high density floppies. We did. It takes 9.75 seconds to assemble HALGOL logical disk #1 if the binary code is saved or 8 seconds flat if not. (It turns out that one often just wants to leave the object code in memory for the time being.)

463 seconds to 53.8 seconds to 9.75 seconds. Interesting!

Oh, yes: the real DOS. Well, we chose to configure it as a single disk system with the single disk configured as a system disk with two catalog volumes. Track #0 (8K) is used for disk catalogs for volume #3 (4K) and volume #4 (4K). Tracks 1-8 inclusive (64K) are reserved for the HALGOL code for booting (HALGOL now

After 12 years as a programmer, Maxine realizes the excruciating truth: she is no longer excited by hard copy

FORTRAN PROGRAMMERS

Looking for the right PC FORTRAN LANGUAGE SYSTEM? If you're serious about your FORTRAN programming then you should be using F77L- LAHEY FORTRAN.

Editor's Choice - PC Magazine

- Full FORTRAN 77 Standard (F77L is not a subset)
- Popular Extensions for easy porting of minicomputer and mainframe applications
- COMPLEX*16, LOGICAL*1 and INTEGER*2
- Recursion - allocates local variables on the stack
- IEEE - Standard Floating Point Arithmetic
- IMPLICIT NONE
- Long variable names - 31 characters
- Fast Compile - Increase your productivity
- Source on Line Debugger (Advanced features without recompiling)
- Arrays and Common Blocks greater than 64K
- Clear and Precise English Diagnostics
- Compatibility with Popular 3rd Party Software (i.e. Lattice C)
- Easy to use manual
- Technical Support from LCS

F77L - THE PROGRAMMER'S FORTRAN

$477.00 U.S.

System Requirements: MS-DOS or PC-DOS, 256K, math coprocessor (8087/80287)

FOR MORE INFORMATION: (702) 831-2500

Lahey Computer Systems Inc.
P.O. Box 6091 Incline Village, NV 89450/U.S.A.

International Dealers:

England:	Grey Matter Ltd.,	Tel: (0364) 53499
Denmark:	Ravenholm Computing,	Tel: (02) 887249
Australia:	Computer Transitions	Tel: (03) 537-2786
Japan:	Microsoftware, Inc.,	Tel: (03) 813-8222

SERVING THE FORTRAN COMMUNITY SINCE 1967

Circle Number 44 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

occupies just over 40K of object code). Tracks 9–43 inclusive (280K) comprise volume #3, which we are using to hold the HALGOL programs, mostly for assembly, etc. Tracks 44—159 inclusive (928K) comprise volume #4, which we are using to hold the HALGOL source code. With everything loaded up, the disk is about half full!

There is a simple BACKUP program which produces an identical copy of the 1.28Mb floppy. This program takes 32.6 seconds to read the source disk. Then the destination disk is placed in the drive, and it takes another 32.6 seconds to write the disk. Add another 55 seconds if the destination disk needs to be formatted. Needless to say, one needs a 2 Mbyte Beaucoup Grande to do this!

HALGOL Release v.04:

We (James and I) have adapted the current version of HALGOL, including both QD DOS and the real DOS for the high density floppy, to the IBM PC. We have decided to freeze HALGOL at this point and call it v.04. All we need is some documentation, and we will send a mailing to all registered Grande or Beaucoup Grande owners in the U.S. announcing its price (which will be based on the number of pages of the documentation and the number of disks) and availability. Our guess is that the documentation will run about 50 pages (v.03 had 28 pages) and that the Apple version of the HALGOL source code, which is optional, will fit on six disks. Another two or three disks will be needed for the Apple II version of the object code, utilities, demos, and the new resident assembler. (The source code for the assembler will not be made available.)

The IBM version will need only three disks to hold the source and one or two for the rest, so the IBM version will be cheaper. (The whole damn shooting match—source code, utilities, demos, *everything*, fits on a single 1.28Mb floppy with some room left over!)

Since I do not want to be sentenced to hard labor in a federal prison, we will not sell software outside the U.S. If you are a foreigner who has obtained a Grande, I suggest you cultivate a U.S. pen pal.

Death of a Host?

The Apple II is about to be history. The failure of Apple Computer to provide a real keyboard, legible 80-column upper/lower case, and real floppy disk drives was undoubtedly based on compatibility. Sigh. When you are compatible with a donkey cart, what you have is a donkey cart—a point that neither Apple Computer nor many of my readers seem to understand. Our future emphasis will be on the use of the PC as a host for the DTACK boards, not the Apple II. If you have a Grande or B.G., and you also have an IBM PC, then it will cost $100 for an interface, cable, documentation, and disks to be up and running HALGOL v. 04 and QD DOS in a real-computer environment.

Just about everybody who builds widgets to plug into host computers has switched from the Apple II to the IBM PC for the same reasons we are doing so. Too, HALGOL as a language has simply outgrown the Apple II.

Benchmarks:

Most of you have heard of the "Whetstone" benchmarks for floating point performance. The Whetstone performance rating gives a pretty good indication of a computer's suitability to do heavy duty scientific calculations. However, there are lots of appli-

object $49 — BASIC WINDOWS — asm source $149 *

Create "instant" pull down menu's or pop-up help windows using any color combo's desired. Basic Windows uses a smart window buffer manager which does not use available BASIC memory! Also allows async operations for a dozen simultaneous windows. Screen save/restore, window create/restore/toss, window print using word wrap & other controls, Input string$ from window, clear/scrollup/ scrolldn/clearEOL&EOS. Includes 2.5k code, doc, test program. For IBM & Compaq BASICA or IBM & MSQUICK compilers on Dos 2 or 3.

object $59 — C DATA ENCRYPTION — c source $249 *

A complete security system using DES (Data Encryption Standard) per U.S. government standard FIPS PUB46 written in Microsoft C. The 3 modes are supported: Electronic code book, cipher feedback, and cipher block chain. Also included is data compression and telecommunications formatting for transmission and storage via any network medium. Send your electronic mail encrypted via Telemail, MCI mail, Easylink Telex, or BBS ?!!

object $99 — COMMX — consolx $69 option

COMMunications eXchange provides multi-terminal emulation, multi-protocol file transfers, unattended control macros and multi-site dial & parameter directory. Emulates VT100/102/52, IBM3101-1x, WYSE50, TV950, HP2624/45, ADM3/11, ADDS R60 (all character modes). Terminal modes support full color controls with optional 25th status line showing time and caps/num lock status. File XFER for ASCII, KERMIT, XMODEM, and COMMX (faster than KERMIT using COMMX/Mainframe). CONSOLX option provides remote PC operation.

HAWKEYE GRAFIX Inc
3415 Hyde Park Drive
Clearwater, FL 33519
call => 813-786-8161

* discounted thru Sept 1986

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

cations where floating point is mostly not required, such as software development, word processing, and accounting. Somebody has therefore invented a "Dhrystone" benchmark, which is intended to measure the combination of CPU and compiler non-floating-point performance. The benchmark also ignores disk and other I/O performance figures. Apparently the only compilers to be benchmarked are C compilers.

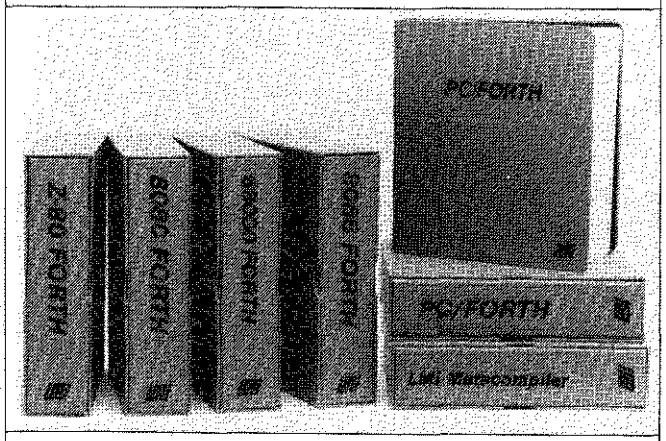
Malcolm Slaney has sent me a copy of about 150 Dhrystone benchmarks taken off Arpanet, which were apparently compiled by Rick Richardson. While quite a few of the results are for micros, wait-states were not specified. I am going to reprint a few of the results, giving the *best* performance when a range is given (better C compiler or fewer wait-states):

	Amdahl 5860	UTS/V 5.2	28970
	Cray X-MP/48	CTSS	17857
	IBM 3083	VM/CMS HPO 3.4	13889
	VAX 8600	VMS	7142
(1)	NCR Tower 32	SYS 5.0 Rel 2.0	4545
	IBM 4341 M.12	UTS 5.0	3910
(2)	WICAT PB	System V	2233
	AT&T 3B20	UNIX 5.2	1724
	VAX 11/780	UNIX 4.3 bsd	1662
	MicroVAX II	–	1612
(3)	IBM PC AT	MS DOS 3.0	1388
	AT&T 3B2	UNIX 5.2	1315
(4)	SUN 2/120	Standalone	1315
(5)	DSI-32	MS DOS/Greenhills	1315
	PDP 11/70	UNIX 5.2	1250
	VAX 11/750/FPA	VMS	1091
(6)	NSC ICM-32016	UNIX SVR2	1084
(7)	Atari 520 ST	TOS, DRI	846
(8)	Macintosh	Megamax C 2.0	709
(9)	IBM PC	Aztec C v3.2d	454
(10)	Amiga	Lattice 3.02	371
	HP-110	Lattice 2.14	284
	C-64	C-Power 2.8	36

NOTES: (1) is a 16.67MHz 68020 machine (rating 4545) that blew all the other microprocessor-based machines out of the water, except for other 16.67MHz 68020 machines that crowded in right behind, performance-wise. The closest microprocessor-based competition that was not another 68020 was (2) WICAT's PB (proprietary bus) 12.5MHz 68000 system (rating 2233)! The closest non-680X0 microprocessor-based competition was AT&T's high-priced 3B20 (*not* the same as the cheaper 3B2!) (rating 1724) and then, finally, a hopped-up 9MHz 80286-based PC AT (rating 1556). The vanilla PC AT rated 1250. The Nat Semi 10MHz 32032-based DRI board, which runs in a PC host environment, crawled slowly under the wire at 1315 even using the fabled Green Hills C compiler. The 10MHz 32016 gasped in at 1084.

MORE NOTES: (3) was the fastest stock (6MHz 80286) IBM PC AT. (4) is a 10MHz 68010. (5) is the 10MHz 32032 attached processor board that was written up in *BYTE*. (6) is a 10MHz 32016. (7) is a stock Atari 520 ST, (8) a stock Macintosh, (9) a stock 4.77MHz 8088-based IBM PC, and (10) a stock Amiga. Judging by the fact that Wicat got a 12.5MHz 68000 to rate at 2233, I would guess that the 520 ST, Mac, and Amiga are all suffering from inefficient software. Remember, these benchmarks are for CPU/C compiler combos.

TOTAL CONTROL with *LMI FORTH*™



For Programming Professionals:

an expanding family of compatible, high-performance, Forth-83 Standard compilers for microcomputers

For Development: Interactive Forth-83 Interpreter/Compilers

- 16-bit and 32-bit implementations
- Full screen editor and assembler
- Uses standard operating system files
- 400 page manual written in plain English
- Options include software floating point, arithmetic coprocessor support, symbolic debugger, native code compilers, and graphics support

For Applications: Forth-83 Metacompiler

- Unique table-driven multi-pass Forth compiler
- Compiles compact ROMable or disk-based applications
- Excellent error handling
- Produces headerless code, compiles from intermediate states, and performs conditional compilation
- Cross-compiles to 8080, Z-80, 8086, 68000, 6502, 8051, 8096, 1802, and 6303
- No license fee or royalty for compiled applications

For Speed: CForth Application Compiler

- Translates "high-level" Forth into in-line, optimized machine code
- Can generate ROMable code

Support Services for registered users:

- Technical Assistance Hotline
- Periodic newsletters and low-cost updates
- Bulletin Board System

Call or write for detailed product information and prices. Consulting and Educational Services available by special arrangement.

Laboratory Microsystems Incorporated
Post Office Box 10430, Marina del Rey, CA 90295
Phone credit card orders to: ***(213) 306-7412***

Overseas Distributors.
Germany: Forth-Systeme Angelika Flesch, Titisee-Neustadt, 7651-1665
UK: System Science Ltd., London, 01-248 0962
France: Micro-Sigma S.A.R.L., Paris, (1) 42.65.95.16
Japan: Southern Pacific Ltd., Yokohama, 045-314-9514
Australia: Wave-onic Associates, Wilson, W.A., (09) 451-2946

Circle Number 63 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

This Arpanet listing carried about 149 benchmarks of computers that were *not* Atari STs. Here is what the price of some of these machines would have to be to have a cost-effectiveness rating of 1.06 like the ST:

Cray X-MP/48	17857/1.06	$16,846
IBM 3083	13889/1.06	$13,103
Cray-1A	13888/1.06	$13,102
VAX 8600	7142/1.06	$6,738
IBM 4341 M12	3910/1.06	$3,689
Pyramid 90X	3333/1.06	$3,144
VAX 11/780	1662/1.06	$1,568
IBM PC AT	1388/1.06	$1,309
Macintosh	709/1.06	$699
IBM PC	454/1.06	$428
Amiga	371/1.06	$350
C-64	36/1.06	$34

The reason the computer industry is largely hostile to measures of performance is that if you benchmark 100 computers, 99 of them are *not* going to finish in first place! Since all 99 of the companies that do not finish in first place believe that they have a right to stay in business, *you* therefore have a responsibility to make purchase decisions based on factors other than performance! Uh, that's right, isn't it?

Applicability:

Some of the most ludicrously inappropriate suggestions emanate from the full-time computer professionals in the UNIX market, which is why I enjoy writing about UNIX and UNIX people. The latest idiocy comes by way of Howard Sachs, who finished second to Seymour Cray in defining the architecture of the next Cray machine and so is now at Fairchild, where he headed the design team for the Clipper. At the recent Uniforum conference in Anaheim, CA, Howard and a bunch of other full-time computer professionals, mostly UNIX types, noticed that there was a lot of really neat software which ran under PC DOS but not under UNIX. Howard had an "obvious" fix for the problem: "'The best way to go would be to rewrite applications like Lotus 1-2-3 in the C language, then the best microprocessor for the job can be chosen,' noted Sachs on a more serious note." (*EET*, 10 Feb, p. 20). *Ye gods and little fishes!*

Howard Sachs, who is a very smart man in his own field (arena of applicability), has just proved that he knows as much about the personal computer marketplace as I know about brain surgery! He obviously has *no idea whatever* why 1-2-3 has rung up hundreds of millions of dollars in sales to date!

Hewpie's Risk:

Just as RISC/RISK has become a cliche, it became a cliche about a year ago that Hewlett-Packard was "betting the company" on its new RISC machines. Well, sometimes cliches are true. By replacing its cash cow, the 3000 series, with a software-incompatible line of CPUs, one which is based on a yet-unproven assertion, Hewpie has taken an enormous, uh, *RISK*! (Sorry about that!)

The rumor is that there has been a shortfall in the actual as opposed to theoretical performance of Hewpie's new RISC machines. A performance increase of 5 had been expected with respect to the old 3000 series. Rumor says the actual measured performance increase is between 2 and 2.5 to 1. And that is *not* good, considering that a routine re-design of the 3000 series using current technology would, the rumor goes, produce a performance increase of 3 *while remaining software compatible* with the existing line!

There is increasing evidence that the theory underlying performance projections of RISC architecture does not correspond with the real world on a one-to-one basis: For instance, a relatively inexpensive 68020 CISC machine (the NCR tower 32) whipped a Pyramid 90x RISC machine (typical price: $500,000!) by 36.4% in the Dhrystone benchmark. Another for instance: The theory underlying RISC projects memory requirements 50% greater than equivalent compiled code on a CISC machine (VAX), but *actual* memory requirements were 250% (!) greater (Sept '85 IEEE *COMPUTER*, p. 14).

Theory is wonderful when it corresponds one-to-one with the real world. *Theory which does not correspond with the real world, however, is* LESS *than wonderful.* Discovering that lack of correspondence can be very painful, especially if you are a conservative manager who has "bet your company" on a theory. . .

68881 Math Chip Report:

There isn't one. While Motorola tech reps will speak lyrically and at length about the 68020, when you ask about the 68881 they suddenly begin to mumble unintelligibly and then they remember an urgent appointment for which they are already late. This means either that there has not been a new mask since the one back in August or else that there *was* a new mask, but it flopped. My guess is that there hasn't been a new mask because the memory management chip for the 68020 was given priority. Remember, there is an industry-wide shortage of skilled hands. And if you assume that the 68020 is a chip for a multi-tasking environment, then memory management *is* more important than a floating-point accelerator! My particular interest in the 68020 is for a high-performance single-tasking personal computer (or attached processor) that does *not* need memory management but *does* need a floating-point accelerator. Ah, well. Meantime, a bunch of would-be customers keep asking us why *we* can't ship the 68020/68881 since lots of other companies claim to be doing so. . . □

—Hal W. Hardenbergh, President, Digital Acoustics Inc.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LUCID, INC.

Learning UNIX/C In-Depth

"FOR THE CLARITY YOU EXPECT FROM TRAINING"

Ask about our HANDS-ON course for "C" Program Development Under MS/DOS"

For Free Brochure, Information or Registration Please Write or Call:
Alice Moss — Director

LUCID, INC.

260 Fifth Avenue
New York, N.Y. 10001
Suite 901

(212) 685-7106

Unix is a trademark of AT & T Bell Labs
MS-DOS is a trademark of Microsoft Corp.

Circle Number 59 on the Reader Service Card

Lilly Siltwell, DP Manager, encourages staff members to do regular back-ups. OR else.

PRODUCTIVITY TOOLS

From Opt-Tech Data Processing

NEW Version **Opt-Tech Sort™** NEW Version

ALL NEW Version 3.0 features even faster sorting, record selection, output record reformatting, dBASE III files, comma delimited fields, and much more. This high performance sort/merge/record selection utility can be used as a stand-alone program or called as a subroutine from most languages.

Supports unlimited filesizes, multiple input files and fixed or variable length records. Many special file types are supported including Btrieve and dBASE. Up to nine sort control fields (ascending or descending), all common data types supported. Output files can be combinations of full records, keys or pointers, subsets of the input file fields, and literal values.

Written in assembly language for **high performance.** Example: 4,000 128 byte records sorted to give key and pointer in 30 seconds. **$149.**

On-Line Help™

A comprehensive utility for adding help windows to your programs. It provides efficient utilities and routines for interfacing your programs with the help system routines and help message libraries.

Help windows are displayed in a fraction of a second. You have total control over the contents of the window, its size and its position on the screen, including the display and border colors.

On-Line Help can be interfaced with interpreted Basic and all popular compilers. **$149.**

Scroll & Recall™

Allows you to conveniently scroll back through data that has gone off the top of your display screen. Up to 27 screens of data can be recalled or written to a disk file (great for documenting systems operations).

Allows you to easily recall and edit your previously entered DOS commands and data lines without re-typing.

Scroll & Recall is very easy to use. It's a resident utility that's always there when you need it! **$69.**

All programs IBM PC/XT/AT & MS-DOS compatible.
Visa, M/C, AMEX, Check, Money Order, COD or Purchase Orders accepted.
Quantity and Dealer Discounts Available
To order or to receive additional information write or call:

Opt-Tech Data Processing
P.O. Box 678 - Zephyr Cove, NV 89448
(702) 588-3737

Circle Number 16 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Domestic and Overseas Report

Frank D. Greco

Parallel Computers, RISC, UNIX, and News From Across the Pond.

Parallel Processors Developed in Parallel

The $14 billion ITT Corporation and two-year-old startup Ncube Corporation (Beaverton, Oregon) are both developing boards that will convert the IBM PC AT into a multiple-processor supercomputer system. Ncube was founded by three former Intel employees. They are basing their future on implementing a Hypercube-based dataflow parallel computer. The Hypercube is based on the Cosmic Cube architecture developed at California Institute of Technology. Ncube's board will contain 4 Ncube processors (nodes), and up to 4 Ncube boards can be plugged into an IBM PC AT. Each board, says Ncube, will boost floating point performance to 20 times that of the PC AT with an 80287 coprocessor. To control the system, Ncube developed two different operating systems that run in parallel: AXIS, which is based on UNIX, will run on the 80286 and handle I/O and the user interface; and VERTEX, which runs on the Hypercube nodes. Each processor contains 160,000 transistors and a cpu that handles 8-, 16-, and 32-bit operations and has on-chip 32- and 64-bit floating point units.

Cray-zee Seymour, His Megaflops are Insane. . .

Similarly, Shelton, Connecticut's ITT is focusing on creating a Personal Supercomputer Coprocessor that will provide peak performances of about two-thirds of a Cray I supercomputer. ITT's board will debut in 1987. They are saying that this board will handle variable word sizes and deliver greater than two billion 16-bit operations per second. 32-bit floating point performance should be an astounding 120 Megaflops! Its cost? About $10,000. That's not bad for a baby Cray!

It should be interesting to see the effect of these two upcoming expansion boards on the burgeoning workstation market (Sun, DEC, Apollo, IBM RT).

"Cray-ving" for More Power

According to the chief executive officer of Cray Research, Seymour Cray, the Cray 3 will be in prototypical form sometime in 1987 and will be commercially available in 1988. The machine will be outfitted with 8 Gigabytes of RAM (that is, one billion 64-bit words) and will feature 16 parallel processing units. And the main processing unit will be the size of today's IBM PC.

BRISC Chips

The British-built Inmos Transputer (see "Redirections," *PJ*, Vol. 2, No. 4), a RISC-based (BRitish RISC) microprocessor with 200,000 transistors on chip, is now available in production versions. The IMS T-414 32-bit Transputer is designed for use in parallel processing computer systems. According to Inmos, from a hardware level the T-414 is 10 times faster than Intel's 80286 and 4 times faster than Motorola's 68020. They claim that in floating point operations the T-414 Transputer is 15 times faster than the 68020 and 25 to 50 times faster than the 80286. The T-414 has *no* floating point hardware in the conventional sense, but a version of the T-414 with floating point hardware on chip T-424 is now being developed. Of course, Inmos did not mention which clock versions of the 80286 and 68020 they were benchmarking against; however, comparing conventional chips such as the 80286 and 68020 against the Transputer is like comparing apples with pi-mesons. The Transputer also has the ability to be combined with many other Transputers to create an even more powerful system. It is conceivably possible to build a supercomputer the likes of the Cray XMP giant by combining many Transputers in parallel at a cost of about several hundred thousand dollars (about the same as a VAX 11/785). Look for a Transputer-based board for the PC to be available in about a year; I hear that at least three vendors are working on such a beast.

Floating Point Systems, Inc., has engineered such a Transputer-based super-

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

mainframe. They have taken many Transputers and linked them together to create inexpensive powerhouse computer systems. In *Business Week* magazine, executives from Floating Point said that the top-of-the-line Transputer based system could theoretically perform an amazing 262 GFLOPS; that's 262 *billion* floating point operations per second. Since the reigning champ in the supercomputer arena, the Cray XMP-2, peaks at 2-3 GFLOPS, look for a new leader because the high-end Floating Point machine tops that by a factor of about 100!

Just When You Thought It Was Safe to Get Back into Programming . . . MicroVAX III

DEC is ready to announce an updated version of the MicroVAX supermicro tentatively called the MicroVAX III. The MicroVAX III, unlike the MicroVAX II, emulates the full VAX instruction set (over 300 instructions) on 8 VLSI chips (the older VAX 11/780 uses over 3,000 IC chips). It should have a virtual memory capacity of 4 Gigabytes and operate at 1–2 MIPS and cost around $20,000. Rumors circulated at the latest DEC-Expo has DEC beta testing a RISC version of a desktop VAX at selected sites.

Do This One for the Clipper

Fairchild's General Manager of Advanced Processor Division Howard Sachs has led a development team to engineer a super high-performance 32-bit microprocessor that achieves peak rates of 33 MIPS (although its average processing is rated at "only" 5 MIPS—5 times a VAX 11/780!). This 3-chip set (846,000 transistors) is called the Clipper and includes on-board floating point processor and 2 on-board memory management/cache units. Similar to some other, newer, chip designs, the Clipper is RISC-based. Although not a pure RISC chip set because it has over 100 instructions and some of these instructions take longer than one clock cycle, the Clipper does have its instructions hard-wired rather than microcoded. In addition, the Clipper can add new functions through a 2K instruction macrocode ROM that can create more complex operations from the basic hard-wired set. Sample cost is only $2,450 per module (this module measures 3 inches by 4.5 inches!)

Sachs has said that the Clipper is designed to work with popular peripheral chips. He said ". . . the Clipper can use a processor such as the 80286 as an I/O processor." He also said that he expected the cost per board to be about $1500 at the end of 1986. A Sun Microsystems engineer told me recently that Sun is very interested in this chip set for high speed graphics and that Sun has a sample board in their labs already, along with a couple of Intel 80386s. Imagine a PC AT BOARD (!) with the 5 MIPS Fairchild Clipper and UNIX for under $5,000!! Number Crunching? How about Number Vaporizing!!!

No-RISC Insurance

Speaking of RISC chips, Motorola and Intel are both busily working on new RISC-based microprocessors. I haven't found out what the Intel chip is tentatively called (yet), but the Motorola chip is slated to be called the 68032 and is scheduled to be available in 1989. The phone company is supposedly also working on a RISC processor, "optimized for the UNIX operating system," called the C-Machine. I asked a few AT&T representatives (including some Bell Labs contacts) what they mean by "UNIX optimization," and *no one* knew for sure what that meant (there were a few guesses at function call stack-frame push/pop optimization)! Didn't they say the same thing ("optimized for UNIX") about the WE32000 chip, a.k.a. the BellMAC 32 chip, about 4 years ago?!! Aren't the benchmarks on the AT&T 3B2 model 300 very poor compared to similar supermicros'? Instead of building their UNIX-optimized RISC chip, how about optimizing UNIX itself: correcting some of the deficiencies such as the weak Inter Process Communication, the unportable and unreliable signal handling facilities, the extremely high overhead of context switching and process forking, the lack of shared library code, the small number of concurrently open files per process (20), etc., etc.

Just the Facts Ma'am, Just the Facts

A certain self-proclaimed queen of UNIX has often gotten up on her soapbox and told you what she thought of things whether you liked it or not. Well, last March she predicted that AT&T would be very successful with the UNIX PC and would sell 100,000 units. We all know now that she was wrong; the UNIX PC is a marketing fiasco. AT&T was advertising the thing in the *Wall Street Journal* with full page ads saying UNIX . . . UNIX . . . UNIX. There's a mistake right there. Did you ever see an IBM ad, "Buy our new VM MAINFRAME" or "Buy our PC DOS PC"? Did you ever see a DEC ad, "Presenting our new VMS PC"? The machine should have been presented as the AT&T PC 7300—a Business Solution. In a business periodical like the *Wall Street Journal*, you must sell a business solution to the reader, not some operating systems buzzword. And while AT&T reports that thousands of UNIX PCs have been sold, the vast majority that are "sold" are for internal use. The same goes for their 3B line. It's all paper sales, not external cash flow.

In 1983, I read a report from this guru of UNIX and her staff that the National Semiconductor 16000 chip (now the 32032) is the perfect chip on which to run UNIX because it is so close to the DEC VAX architecture (her words not mine!). She predicted that system manufacturers would flock to National Semiconductor to buy those chips. National Semiconductor has since been involved in some unfortunate problems with the government and has been banned from doing business with the military for awhile. I haven't heard of one major manufacturer jumping on the National Semiconductor 32032 bandwagon. As a matter of fact, those that were once interested are jumping *off* the Nat Semi bandwagon.

In 1984, this UNIX maven was constantly raving about a software product called "Higgins," which increases the usability of a PC. She told me face-to-face that Higgins was going to be an example of what next generation software looks like. In 1985, *Infoworld* gave Higgins a lukewarm reception, complaining that the screen was "too busy" and its price too high. Where is Higgins today?

The September 1984, issue of *Fortune* magazine, reported that she predicted, "Within 18 months AT&T will be conforming to the IBM standard—XENIX". Here it is, 1986—XENIX has been criticized for having thirty pages of bug reports, and IBM shows little sign of helping those who purchased Version 1.0. In addition, contrary to her prediction, XENIX has conformed to the AT&T System V standard (although it has added features lacking in AT&T's System V).

Prognosticators such as this make it rough on the *real* UNIX supporters out there. It is *her* predictions that the MS DOS proponents attack. With statements such as, "I expect a multi-billion dollar market for UNIX by 1986" (1981 or 1982), no wonder the DOS folks are skeptical. I once asked her where she got her market figures, and she told me "intuition" ! That is no lie! The DOS people think that her market projections represent the aggregate feelings of the UNIX market. With crystal-balling like

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

this, it's difficult to convince already-skeptical end-users who read the PC trade press to even *try* a UNIX system.

If Steve Bourne Wrote Another Shell, Would It Be Called the Bourne-Again Shell

AT&T's newest UNIX shell utility, called the Korn Shell (ksh) after author David Korn, is available for a nominal fee via an on-line catalog called the AT&T UNIX System Toolchest. The ksh offers many-"csh" functionality such as a history mechanism, aliases, and several performance increases. It also allows users to define their own built-in parameterized functions, built-in arithmetic, built-in substring capability (the Bourne shell should've had that a *long* time ago — FG), job-control, and one-line visual editing. The number is (201) 522-6900; use the UNIX login "guest."

JAPAN INC.— NEWS AND RUMORS

It's LONLI at the the Top

Hitachi has developed a new language called LONLI, which stands for Logic Oriented Language Inferencer. It's similar to PROLOG but differs in that Kanji capability, floating point processing, and graphics and database operations are built into the language. According to Hitachi, programming productivity is 10 times faster than with PL/I or FORTRAN.

It's NEC and NEC All the Way

Most of you out there know the increased power that the V-Series microprocessors from NEC give the PC owner. Users on the USENET have raved especially about the V30, a replacement for the 8086. A couple of recent postings to the "Net" state that a simple chip replacement in an AT&T PC 6300 increases its power to that of an IBM PC AT. Big deal, you might think, but remember that a 6300 system with a 20 Megabyte hard disk costs in the neighborhood of $2500 while the IBM PC AT can cost almost $2000 more!

NEC is also working on an 80386-compatible microprocessor that has 80387 capability *on-chip*! That means you won't have to access an auxiliary (external) processor, which implies faster calculations and simpler processor-coprocessor synchronization problems to solve. In addition, NEC is already working on a version of this chip (called the V71) that will have 4Kbytes of cache memory *on-chip*! With the high speed cache memory, recently-accessed variables and their neighbors could be kept "local" to the processor instead of in slower RAM memory and could thus be available to any program that accesses these variables.

Does the V in NEC's V-Series Stand for Victory?

What is really interesting about the V-Series processors is their effect, not on Intel, but on IBM. IBM is surely counting on the Intel iAPX386 (the official name of the 80386 because you cannot copyright numbers) as the cornerstone of their next major workstation. Running multiple operating systems such as MS DOS and UNIX systems concurrently has definite advantages. But the problem is that NEC could supply IBM's competitors with a super-powered version of the Intel chip. One might naively conjecture that IBM would just use the NEC chip instead of Intel's. But IBM would surely *not* go with the NEC chip. It's a Japanese chip; just think of the political ramifications! The number one computer manufacturer in the United States would never use a Japanese-produced chip as the *heart* of its machine (peripherals yes, CPUs no). In addition, Intel would definitely bring a lawsuit against major-stockholder IBM. It's a

Atron's PC/AT Bugbusters

Hardware-assisted Software Debuggers for Bullet-proof PC/AT-based Products

A BUGBUSTER STORY

Brad Crain, a project manager at Software Publishing (the people who developed both PFS:WRITE and PFS:FILE), relates the following: "On Friday, March 22, 1985, I was about to get on an airplane with Jeff Tucker, who was co-author of PFS:WRITE with me, and fly to IBM's Boca Raton, Florida facility. For a week, we had been unsuccessfully trying to isolate a bug in a new software product. In a last, desperation move, I set up an early-Saturday morning appointment with ATRON.

"Three of us walked through ATRON's door at 8:00 the next morning. Using ATRON's hardware-assisted debugging tools, we had the problem identified and fixed by 10:30AM."

Mr. Crain concludes: "We'd never have found the bug with mere software debuggers, which have the bad habit of getting over-written by the very bugs they're trying to find. It doesn't surprise me that almost all the top-selling software packages were written by ATRON customers. Now that they've broadened their PC family of debuggers to include a PC/AT debugging tool, those of us seriously into 80286 development are greatly relieved."

ARE YOU TRYING TO DO SOMETHING SCAREY?

Like developing your AT-based software product in the dark? Without professional debugging tools?

Seven of the ten top-selling software packages listed by the *THE WALL STREET JOURNAL** were produced by ATRON customers. The PC PROBE™ bugbuster ($1595) accounts for much of this success. Now that the PC/AT is the new standard for advanced commercial and scientific development, ATRON is proud to announce the AT PROBE™ bugbuster ($2495). It has even more debugging capabilities than the PC Probe.

HOW BUGBUSTERS KEEP YOU FROM GETTING SLIMED

The AT PROBE is a circuit board that plugs into your PC/AT. It has an umbilical which plugs into your 80287 socket and monitors all processor activity.

Since AT PROBE can trace program execution in real time, and display the last 2048 memory cycles, you can easily answer the questions: "How did I get here?" and "What *are* the interrupts doing?"

It can solve spooky debugging problems. Like finding where your program overwrites memory or I/O - impossible with software debuggers.

You can even do source-level debugging in your favorite language, like C, Pascal or assembler. And after your application is debugged, the AT PROBE's performance-measurement software can isolate your application's bottlenecks.

Finally, the AT PROBE has its own 1-MByte of memory. Hidden and write-protected. How else could you develop that really large program, where the symbol table would otherwise demand most of your PC/AT memory.

BORLAND'S PHILIPPE KAHN: "THERE WOULDN'T BE A SIDEKICK™ WITHOUT ATRON'S DEBUGGERS."

So why waste more time reading though your program listing for the ten thousandth time, trying to find why your program starts howling with every full moon. Be like BORLAND, get your Atron bugbuster today and bust bugs tomorrow.

atron
THE DEBUGGER COMPANY
20665 Fourth Street • Saratoga, CA 95070 408/741-5900

*WSJ, June 24, 1985, reporting Softsel figures. © 1985 by ATRON. PC PROBE™ and AT PROBE™ ATRON. SIDEKICK™ Borland. IBM Corp. owns numerous trademarks. Ad by TRBA.

Circle Number 1 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

serious problem for IBM. If I were Intel, I would be trying my best to convince my ally in Armonk's legal beagles to lend a hand in the Intel/NEC legal embroilment.

TRON on NEC's V Series Processors

TRON, The Real-time Operating-system Nucleus, is a new operating environment developed by Ken Sakamura who is a computer science professor at the University of Tokyo. Sakamura believes that the most widely used computers of the future will be high-performance personal workstations. He feels that these "superpersonals," as he calls them, must be able to service a spectrum of real-time functions. With that in mind, he built an OS nucleus from which three different types of operating systems could be built. The three basic types were industrial, business, and environment control. This results in the three names: I-TRON, for engineers developing embedded systems and requiring fast task dispatching (task switching on the order of several milliseconds); B-TRON, for Business users; and H-TRON, for environmental control in the smart Home (or office or factory) of the future. All three have C language interfaces.

The business version, B-TRON, has expanded debugging functionality, bit-mapped display management, and parallel processing hooks. H-TRON heavily emphasizes the man-machine interface and will include interfaces for communications, music, audio, and electrical-device switching. I-TRON will be used in industrial applications for controlling devices in real-time.

The ultimate goal is to put TRON in hardware and place it on-chip with a 32-bit CPU. A personal computer prototype was built with a Motorola 68010 CPU and a bit-pad (instead of a mouse). The codesize of I-TRON on a 68K machine is only about 6-8K. Sakamura says that a commercial version of a software-only TRON machine will be available in the middle of 1986. A hardware TRON is about 2 years away. NEC is implementing this OS nucleus on the V20 to the V60 processors and is calling the software RX116. Fujitsu has it implemented on the iAPX286 family and calls it MMU286 OS. Hitachi, second source of Motorola's 68000 family, will market their version ITOS68K in June of 1986. Don't you just love those colorful sounding names?. . .

Please Tell Me Again and Again

At Tokyo Industrial College, Professor H. Enomoto has developed a general purpose software development system with a natural language front end. It's called the TELL/NSL system and incorporates inference, association, and learning. Supposedly, if you describe a given software system to TELL/NSL over and over again, it learns and infers about all of it and starts to make reasonable suggestions on enhancement.

If you think that Microsoft's MSX operating system is a bust, think again. According to the Japan Electronic Industry Development Association, 83.3% of all low-end microcomputers in Japan run MSX, which is about 80,000 units sold per month.

Former Ichi-ban Nishi-San Goes Sayonara

Microsoft's MSX father and Tandy Model 100 inventor Kay Nishi has severed ties with Bill Gates' Group. No reason was given. It has been rumored that Nishi felt that the 68000 would predominate on low-end home computers, much to the chagrin of Chairman Gates.

Toshiba's New Floppy Has Its Ups and Downs

Toshiba has developed the first 3½ inch floppy disk drive that uses *vertical* recording techniques. The little cartridge can store up to 4 Megabytes. Toshiba of Japan says quantity shipping is planned for mid-1986.

Toshiba Makes a Spectacle

Toshiba has also just started shipping a 5¼ inch optical disk drive that can store 800 Megabytes. What's interesting about this $8,000 disk drive is its access time, 120 millisecond average. This is faster than a floppy disk drive and almost as fast as the IBM PC XT's 85-millisecond winchester. Times are a-changin'. It wasn't too long ago that optical disks had access times in the neighborhood of 1-2 seconds. It won't be long before the access times are close to voice-coil-driven winchester hard disk drives (30-40 milliseconds).

Thanks for the Memories

Toshiba recently completed construction of an ULSI (Ultra Large Scale Integration) research laboratory and has started activities. One of their first projects is to create a prototype of a 16-Megabit DRAM memory chip. They are also developing the next generation semiconductors, using faster-switching Gallium Arsenide.

Spaghetti with Soy Sauce

Toshiba has just acquired 20% ownership of Olivetti Japan (previously fully owned by Olivetti). Toshiba is obviously looking to expand their market to Europe. With an impressive line of letter quality, multi-font, 24-pin P-series printers (P351, P321), phenomenal 512K MS DOS LCD laptops (I've heard street prices of $1700 for them already), 1 Megabyte RAM chips already in mass manufacture, an MSX2 compatible CD-ROM (600 Megabytes), large deals with Sun Microsystems and Japanese ally Sord Computers, and a new marketing look, I would keep an eye on Toshiba.

Look Sharp

Sharp Electronics of Japan announced two 32-bit graphics workstations. They are based on multi-processor architectures that use up to four 68010 or 68020 microprocessors. Other features include minimum RAM size of 2 Megabytes, high resolution monochrome monitor, and a Japanese version of UNIX.

Cameras, Copiers and Now Computers

Ricoh has made an interesting announcement with their TS3 intelligent graphics terminal. According to Ricoh, they are initially gearing it for the AT&T 3B2 supermicros (I'm not quite sure what they mean by that). It has a 15-inch, bit-mapped screen with a resolution of 1,024 by 768. It has a maximum of 16-plane video memory (that means 65,536 concurrent colors or monochrome shades). The screen is memory-mapped with a 2-Megabyte frame buffer that allows panning and scrolling. And—very similar to AT&T's patented (!!) BLIT terminal—the TS3 allows programs to be downloaded to itself and run locally with minimal host intervention. Unlike the BLIT terminal's, the TS3's display is available in either landscape or portrait mode.

Quote of the Month

This is from some unknown Value Added Reseller (VAR): "We are just as likely to get a paperless office as we are to get a paperless bathroom." □

Readers may write to Frank Greco

c/o Programmer's Journal
P.O. Box 30160
Eugene, OR 97403

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Efficient Custom Programming

by Paul Barkley

Last issue I began a discussion of custom programming practices that I feel are necessary for the survival of the independent consultant/programmer. Since virtually every independent consultant that I know also writes code to make a living, this is an issue that merits some attention.

One of the keys to efficient production is specialization. Most consultants not only specialize in certain vertical markets, but they also specialize in certain languages or systems. There's certainly nothing wrong with a broad background, and in fact this is highly desirable, but unless you are proficient in a particular language, you'll waste a considerable amount of time coding. This translates into either excessive hourly charges for your clients or a lot of hours spent working for free on fixed fee contracts. Specialization will allow you to maximize your productivity by developing sophisticated tools geared to the kind of systems that you write.

I write most single-user code in dBASE III and compile it whenever possible with Clipper, but my multi-user systems are written in a programmer's data base called LAN:Datacore, which requires a high level language such as Pascal, FORTRAN, C, or Compiled BASIC. I manage to achieve pretty respectable results with good old BASIC. (Interestingly enough, I would no more use interpreted dBASE than I would use interpreted BASIC if I could help it — neither is worth the appalling performance loss.)

In the last issue I discussed the need for an input subroutine capable of handling full screen input, which is an acute problem with BASIC. This module can then be used for all input throughout the system. Compiled BASIC has an $INCLUDE statement that lets you write a program module, such as a password screen, and insert GOSUBs every time you want to call the input routine. Instead of using your word processor or editor to copy the enormous input routine into the program code, you simply insert an $INCLUDE statement where the code belongs, and at compilation that code is "included." All the copious remarks in the subroutine (as in the program itself) that you've inserted to explain what devious idea caused a particular piece of tricky code are expunged at compilation and don't contribute to program size.

This is an illustration of the power of writing reusable modules. Once you've expended the effort to create the input subroutine, it's now available to all programs with little additional effort. Further, each program is much clearer and easier to understand because it's not cluttered with repetitive code. I *know* what a GOSUB 60000 does, and if a user is having trouble with inputs I know exactly where to look —not in the program, but in the subroutine.

Experienced programmers may find this illustration trivial and may have accumulated an entire toolbox full of esoteric algorithms for such things as windows, sorting, device handlers, and what not, but for many programmers who repeatedly start from scratch this may provide some clues to more productive programming.

In the last issue I also described the menu driver which keeps menus consistent throughout a system (or all your systems for that matter) and allows you to easily make changes in the hierarchial arrangement of programs.

Finally, I described a report generator with (virtually) unlimited control breaks and multi-file capabilities. All of these modules, though, lead up to an ultimate (and somewhat unattainable) goal: a true application generator.

In our digital world, we're often accustomed to thinking about issues in a binary fashion: something either is an application generator or it's not. Unfortunately, application generators are something like artificial intelligence, in that they're hard to pin down. My suggestion is that you figure out what you think an application generator should do for *you* and then put together your own. That way, you have no one to blame but yourself if you don't like the system, and you never have to worry about royalties and support!

Let's examine this slippery question of degree. dBASE III has several elements of application generation built in, and these are an excellent starting point. If you wish to write a report that uses no more than two control breaks and only one file, the dBASE report generator is excellent. I've used this for many simple reports. Similarly, the built-in label generator is very handy with its ability to throw sample labels on your printer for alignment.

The usefulness, though, is a matter of degree. It doesn't take long to run into a situation where you need a report that uses more than one file, or has perhaps three control breaks. When that happens, you must code the whole report by hand. After you've coded one report by hand, the inclination is to fiddle with the report headers and page number and date placement, and suddenly the built-in report no longer matches your new "improved" format. The next step is to throw away the simple report and code that by hand, too, so that it matches. Finally, the sophisticated programmer stops using the report generator for any project because he or she knows that the work will probably be redone anyway.

This is a mistake. Instead of discarding the idea of report generation, you should *build* on the system. If the built-in report generator is insufficient, build your own that has enough features to suit the type of work that you do. *Your* idea of appropriate features and *my* idea are probably not the same, because we probably don't work on the same kinds of systems, so use the concept and tailor it to your own needs. Besides, there's an innate satisfaction in getting your computer to do your work for you. When you get tired of writing systems to do your client's work, write a system to do your own work!

The concept is exceedingly simple (but sometimes hard to grasp): Since you know what the finished code should look like, write a program that generates this ASCII code as sequential output. For example, if you want one line of a BASIC program to say,

```
REM This section launches missiles
```

write a program line (also presumably in BASIC) that says,

```
PRINT #1, "REM This section launches missiles"
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

This is tricky only when the code has to contain a quotation mark, and this can be inserted as CHR$(34). There isn't any line of code that you can write in a program that you can't also write as a line of sequential output from a program generator!

Now, let's go back to the idea of degrees of sophistication in report generators and apply it to application generators as well. Remember that you can choose as many or as few of these ideas as you wish, and of course you probably have some great ideas that I haven't considered (why don't you drop me a line?).

A system should have continuity and coherence, so a good starting point is some sort of data dictionary. I use one with LAN:Datacore because I like to control the field names throughout the system to prevent duplicates, and in this particular DBMS the programmer must pack and unpack the file buffers—a tedious and error prone task. In addition, because of the multi-user nature of the system, opening a number of files may be aborted by not getting the rights to one file, in which case the program should detect and announce that and close all the files previously opened in that program.

Start by defining a system control file that contains file numbers and filenames. A second control file contains file numbers, field numbers, field names, and field descriptions. Write a short program that asks you what files you want to open, and with what conditions (read/write intentions and willingness to share). This system program then generates the open and close subroutines and the subroutines to pack and unpack the files—probably several pages of code. All you have to do is grab the code with your editor and start adding to it.

Note that all the subroutines are written at specific addresses (labels or line numbers), so *every* program in your system will behave the same way. You don't need a mini-application generator to do this, but it sure beats coding it by hand with the editor.

While you're at it, write a short program that puts a standard heading on the front of a new program with the program name, date, copyright information, compiling and other instructions, program description, and the programmer's name or initials. In a multi-programmer, multi-user development environment, such as a network where programmers are working on different programs in the same system, using the same system control files, another file with names or initials may be necessary. Use of a log-in password to the application generator will identify the programmer.

The menus are generated from similar system control files that contain program names, menu pages, and position numbers. The generator writes the menu program for you.

Another module that I use extensively is a file maintenance module, and it really doesn't need any help from the application generator other than to be renamed so as to start with the proper system initials to suit my file naming convention. This module is smart enough to determine fields by being passed the file number, and it packs and unpacks the buffers accordingly — not quite as efficient as hard coding these activities, but one program fits all the files on a system this way.

This program presents the user with a choice of activities: add, delete, change, and find records, as well as skip forward, skip backward, go to the top or bottom, change keys, and quit. Thus, wherever appropriate (usually as a utility program to fix data entry screw-ups), this program offers the user a common method of accessing data in any file. I can provide this at virtually no cost (to me) because I don't have to rewrite it each time.

So, with just these particular pieces of an application generator, I can drastically cut the amount of time it takes to create a new application. But what about other features?

Many people use a screen painting program. If you type in your screen and indicate input boxes, the screen painter will create the code to write such a screen and will interface with an input module to set up the boxes as well. This is fairly easy with monochrome screens, but becomes more complex with color screens, where there are many more variables that the screen painter must ask you for.

I created some standard utilities that weren't available with LAN:Datacore but which seemed necessary in any DBMS. They don't require rewriting each time and can simply be copied over from models. These include a utility to add a user (this requires specific procedures for each file in the system), remove a user, change a user's password, and rebuild damaged index trees. The latter program also allows a change in file size, which is important in a system that doesn't have a dynamic file allocation scheme. Some DBMSs, of course, may already have these utilities.

A more sophisticated concept, one that underlies the construction of a "true" application generator, is that it should be smart enough to handle changes in file specifications automatically. If you decide to increase a character field from 10 bytes to 12 bytes, you should be able to make that

NEW! **Break the 64K Data Limit**

THE INSIDE TRACK 3.1

Now updated and expanded! **THE INSIDE TRACK** is a collection of advanced utilities for PC/ XT / AT and compatible programmers. Break the 64K data limit, do **WINDOWS** and much more. Most programs run on ANY MS-DOS machine. You'll receive **80 + programs, a comprehensive manual,** fold out reference sheet with **memory map** and **BIOS calls.** Add our assembler code to your programs with **NO ROYALTIES!** BASIC routines are compatible with Quick BASIC and IBM BASCOM2. **All Source code included!** NOT COPY PROTECTED. Some programs require DOS 2.00 or above.

THE INSIDE TRACK
shows you how to:

- Go **beyond** the **64K** data limit
- Do **fast WINDOWS** in your program
- Display data up to 10 times **faster**
- **Reserve memory** for your use
- Read/write files as **fast** as DOS
- **Control** the keyboard settings
- Much, much, more. . . **only $65.00**

NEW! **Break the Lock with**

PEEKS 'N POKES 3.1

Unprotect for GW-BASIC!

PEEKS 'N POKES has been updated and expanded for DOS 3.10, Quick BASIC and IBM BASCOM2. Learn where to **PEEK, POKE, INP** and **OUT.** Swap monitors, swap printers. **DOS/ BIOS function calls** to use ALL of your machine's features — not what your language limits you to. Add our routines to your programs with **NO ROYALTIES.** We supply you with **60 + sample programs,** a comprehensive manual and all BASIC **source code.** NOT COPY PROTECTED. A few programs require DOS 2.00 or above.

PEEKS 'N POKES
shows you how to:

- **Unprotect** IBM and **MS BASIC**
- **Disable** Ctrl-Break and Ctrl-C
- Read and change the **keyboard**
- Read and change the **monitor** status
- Access **printer** & **RS-232** status
- Find more PEEKs and POKEs
- Much, much, more. . . **only $45.00**

ORDER TODAY! M/C and VISA welcome. Shipping: USA $3.00/order UPS; Canada $5.00/order first class; Overseas $16.00/item airmail. Call for upgrade info. Dealer inquiries welcome.

MicroHelp, Inc.
2220 Carlyle Dr • Marietta, GA 30062
Call Toll Free: 1-800-922-3383
Tech support
and inside GA: 404-973-9272

Circle Number 64 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Vitamin C™
A Healthy Supplement For C Programmers

Perfect Windows, Powerful Data Entry, FULLY Integrated, For Effortless Data Entry Windows!

- Complete input formatting
- Unlimited Validation
- Full attribute control
- Field sensitive application help system
- Multiple virtual windows
- Fully automatic, collision proof overlay and restore
- Print to & scroll background windows
- Animated window "zoom"
- Move, grow, shrink, hide, or show any window
- "Loop function" allows processing while awaiting input

and much much more!

Designed to increase your productivity and help you produce superior applications in dramatically less time! Includes 100% source, tutorial, reference manual, examples, and sample programs.

Specify Microsoft, Lattice, Computer Innovations, Aztec, or Mark Williams. Ask about Unix.

$149.95

NOW... VCScreen !

Our new interactive screen "painter" actually lets you draw your data entry windows! Define fields, text, boxes & borders. Move them around. Change attributes. Then the touch of a button generates C source code calls to the Vitamin C routines! Now, your screen designs will be faster and more pleasing when they are created with VCScreen!

Requires Vitamin C library above. For IBM & compatibles.

$99.95

For Orders Or Information,

(214)245-6090

Creative Programming Consultants
Box 112097 Carrollton, TX 75011-2097
Include $3 ground, $6 air, $15 overnight shipping, $25 if outside USA. Texans add 6% tax. All funds must be in U.S. dollars drawn on a U.S. bank.

Circle Number 40 on the Reader Service Card

change in the data dictionary and have the change ripple throughout the application. All programs that access this file should be changed and recompiled to suit the new specification. Obviously, this requires a system control file that contains program names and files, but this information is readily available from the module we already discussed that writes the file I/O for us! The only tricky part is how to handle screen I/O when a field changes—you can decide whether you want the system to adjust your screens (which can be interesting) or perhaps just flag those screens that now need attention.

By now the benefits of a sophisticated application generator should be apparent, but what are the drawbacks?

First, I despair of ever finding a system that suits more than one programmer in the world. Buying an open system that you can modify yourself is an excellent way to begin, but remember that you're likely to complain about some of the features. You won't find anything on the market that exactly suits you, and that's to be expected.

Next, having a generator adds to the number of files that you must keep; thus, for every client I have not only the source code files and the production files, but I also have a set of generator files. This is a minor nuisance.

Finally, any changes you make to your application generation tools must be carefully thought out and either isolated from previously generated systems or carefully integrated. For example, every change made to the input routine in the last two years was carefully added to all client systems, and updates were sent as necessary. Recently, though, a major revision was made to this routine that makes it slightly incompatible (of course), so now it is segregated from the old systems. This adds to the amount of information you must keep organized. Of course, that's what your computer is for, isn't it?.

Home Office

Because of remodeling, I was out of my office for three months and working from my residence again, and although it was nice on those icy winter days, I was ecstatic to get back into regular space. I'm enjoying the professional space, the occasional hall interactions with people in other professions, and the fact that once again I can find all my files and books.

When I wrote the column on home offices (March/April 1986), I never expected to be back in a home office. It reinforced my sense that there are no easy answers to the home office dilemma, and I advise you to think carefully about your own situation.

For Your Information

I read an interesting consulting book the other day that I can recommend. Although it's not particularly well written, is printed from typewritten originals, and isn't as long as some consulting books, it *does* have some interesting tidbits. *How To Become A Successful Consultant in Your Own Field* by Hubert Bermont is available from The Consultant's Library, P.O. Box 309, Glenelg, MD 21737. Write them for a catalog and price list.

Bermont points out that most potential clients have a very narrow "window" when they're interested in your services. This window may be only one or two weeks long, and if you don't happen to hit the window, you're wasting your time.

One of his solutions is to offer broadly advertised free seminars, where he delivers a useful presentation and collects names (attendees must register to get in). These names then become an excellent prospect list of people with currently open windows.

Bermont thinks sending unsolicited brochures is not a good game plan, both because of the window problem and because of the possibility that they will be poorly received. He draws an analogy to getting a brochure from a new psychiatrist that might begin: "Dear Occupant: I know you're disturbed. Everyone is. Why don't you bring your sick mind over to see me—I can help you." Your reaction to such a solicitation would probably not be favorable, even if you *do* have a problem.

It's interesting to note the number of radio and television ads for local alcohol and drug abuse programs (I hope this trend is more widespread than just Washington, DC) that are directed towards *spouses* and family members of those with problems. Perhaps we consultants should send brochures directed, not at the person with the problem, but at his or her boss. They might say, "Dear Owner: Do you have nitwits in your organization who need a computer to straighten out the mess they're making of your business?" Maybe I'll work on the copy a little before I send it out . . .

Drop me a line with your thoughts, experiences, or questions about consulting. □

Paul Barkley
The Barbican Company
6845 Elm Street
Penthouse One
McLean, Virginia 22101
CompuServe: 74776, 25267

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Finally, there's a major end-user computer event for the West Coast in the Fall.

Exhibitors like what the Personal Computer Faire delivers. New business. New opportunities. A professionally run conference and exposition. The right attendees and exhibitors. The highest return on your exhibiting investment.

Quality Attendees

No matter what the immediate business climate, exhibitors at a Personal Computer Faire know they can always count on a steady flow of show traffic. The impressive quality of this attendee traffic may well be your single most compelling reason for exhibiting.

National marketers, such as AT&T, IBM, Kaypro, Hayes, Sigma, Wang, Borland, NEC, Tallgrass and 3COM have relied on the Personal Computer Faire in San Francisco to help increase their market share, and support their local distributors, dealers and VARs in this hotbed of computer activity.

Regional marketers consider the annual Personal Computer Faire to be an ideal marketing vehicle for building sales, developing qualified leads and winning name recognition among thousands of local computer-knowledgeable prospects.

Value-added Show Benefits

- Hands-On Demonstrations of hardware and software.
- Scores of conference program sessions tailored to the needs of attendees.
- Featured industry speakers.
- Special professional seminars on the latest software.
- Attendees who are *true* prospects — knowledgeable and ready to buy.

Marketing & Promotion

Our unique marketing strategy balances direct marketing, publications, magazine and newspaper advertising, conference sessions tailored to attendees' needs, news conferences, year-round publicity and on-site seminars and hands-on workshops.

You'll experience face-to-face selling as you never have before. Our prospect-producing promo programs will deliver the best in quality and quantity. The Interface Group, Inc. produces the largest computer and communications conferences and expositions in the world. Count on us to deliver what you need to make the sale.

But take action now, before our best booth locations are committed. Contact your local Interface Group representative, or one of the offices below.

The Interface Group, Inc. · 300 First Avenue Needham, MA 02194 · (617) 449-6600
West Coast Office: 1401 Dove Street · Suite 220 · Newport Beach, CA 92660 · (714) 476-2300

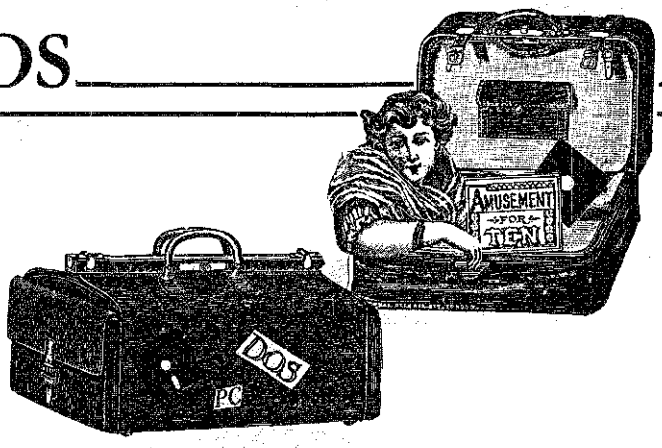
Circle Number 58 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

DOS 2 File Handles and File I/O

by Graham Pearson

This article was developed from a paper presented to a recent meeting of the Houston Area Local Users Group, Assembler SIG.



A Quick Look at DOS 1 File I/O

The following areas must be set up for *each* file in your program before performing any DOS 1 file I/O operation:

1. A File Control Block (FCB), used by DOS for temporary storage of various data associated with a file, such as logical record size, file size, and current record being read or written.
   For the majority of files, the FCB is 37 bytes. For system, read-only, and hidden files, the FCB is 44 bytes. The extra 7 bytes *prefix* the normal FCB area—just to be awkward.
2. An I/O data buffer, whose size is usually the same as the logical record size. Data in this buffer is written to the file during write operations and loaded from the file during read operations.

Straightforward file operations and their DOS function numbers (in hex) are as follows:

Create File: 16H
Open File: OFH
Close File: 10H
Delete File: 13H

There's one trick to the Rename File function, 17H. The second filename (the new name of the file), starting at FCB + 11H, must *not* include a drive code.

Obtaining the file size is not straightforward. Function 23H must be used *before* opening the file, and after setting the logical record size in the unopened FCB, the WORD at FCB + OEH. If the function finds your file, the Relative Record Number, the DOUBLE WORD starting at FCB + 21H, indicates the size of the file in terms of logical records (rounded up if necessary).

Three types of file I/O are available: sequential I/O, random record I/O, and random block I/O. Before performing any I/O, perform the following steps:

1. Use function 23H to determine whether or not the file exists. If it does, any kind of read operation can be performed. Random write operations can also be performed, but usually, sequential write operations assume that a new file is being created. If the file does not exist, the file must be created before any subsequent write operations are performed.
2. Either create the file using function 16H or open the file using function OFH.
3. Set the logical record size in the FCB, the WORD at FCB + OEH.
4. If the file is being opened for sequential I/O, set the Current Relative Record Number in the FCB, the BYTE at FCB + 20H, to zero.

Sequential I/O

1. Set the Disk Transfer Address (DTA) to the address of your I/O data buffer, using DOS function 1AH.
2. Perform sequential I/O using DOS read function 14H or write function 15H.

Random Record I/O

1. Load the Relative Record Number in the FCB, the DOUBLE WORD starting at FCB + 21H.
2. Set the DTA to the address of your I/O data buffer, using DOS function 1AH.
3. Perform random record I/O using DOS read function 21H or write function 22H.

Random Block I/O

This type of I/O is not commonly used. It is similar to random record I/O except that several records are read or written during one operation. Your I/O data buffer must be correspondingly larger. Use this type of file I/O when speed is essential.

1. Load the Relative Record Number in the FCB (DOUBLE WORD) with the first record to be read or written.
2. Set the DTA to the address of your I/O data buffer, using DOS function 1AH.
3. Load register CX with the number of records to be read or written.
4. Perform random block I/O using DOS read function 27H or write function 28H.

Two More I/O Functions

Two commonly-needed file I/O functions not obviously accounted for are "adjust random access file size" and "flush buffer contents."

Adjust Random Access File Size

Files used for random I/O, particularly those involved in database applications, have a habit of getting larger and larger. If a bunch of database records are deleted and the file is "degassed," the only way to downward-adjust the size of the file is as follows:

1. Load the Relative Record Number in the FCB (DOUBLE WORD) with the number of "active" records in the file.
2. Make sure the logical record size in the FCB, the WORD at FCB + OEH, is set. The resultant file size is this value multiplied by the number of "active" records.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

3. Zero out the register CX.

4. Make special use of the DOS random block write function 28H.

Flush Buffer Contents

Each DOS call to perform file I/O does not necessarily result in a physical disk I/O. This is because DOS keeps a data buffer for each file. This buffer is located between your I/O data buffer and the disk.

So what's the big deal about flushing buffer contents? The big deal is this. Buffer contents are automatically flushed when you close a file, but if the "lights go out" in the middle of some database updating process, you could end up with a corrupt database index file, resulting in a total scrambling of your database.

There is one way to ensure physical disk I/O for every DOS file I/O call. This is to use the statement

```
BUFFERS = 0
```

in your CONFIG.SYS file. But this has the undesirable effect of making *all* file I/O rather inefficient.

A better solution is to cheat! After each write operation, we can close the file, using DOS function 10H.

This appears to do nothing more than flush the contents of the DOS buffer to disk and update the disk directory entry—file size, date, and time are updated. The FCB contents are not affected in any way.

Further file I/O proceeds without error using the same FCB. In other words, a file can be closed as often as needed. There's no need to re-open the file or reconfigure the file's FCB to perform subsequent I/O operations.

Summary of DOS 1 File I/O Functions

Create File:	16H	Rename File:	17H
Open File:	OFH	Get File Size:	23H
Close File:	10H	Resize File:	Special use of 28H
Delete File:	13H	Flush Buffer:	Use Close File 10H

Sequential I/O:	Set DTA:	1AH	Read:	14H	Write:	15H
Random Record I/O:	Set DTA:	1AH	Read:	21H	Write:	22H
Random Block I/O:	Set DTA:	1AH	Read:	27H	Write:	28H

DOS 2 File I/O

File I/O is generally simpler using DOS 2 function calls. The following areas must be set up for *each* file in your program before performing any DOS file I/O operation:

1. A File Handle Area. This is a WORD used to save the file "handle" returned by DOS during a File Open or File Create operation.

2. An I/O data buffer, whose size is usually the same as the logical record size. Data in this buffer is written to the file during write operations and loaded from the file during read operations.

Straightforward file operations and their DOS function numbers (in hex) are as follows:

Create File:	3CH
Open File:	3DH
Close File:	3EH
Delete File:	41H

NOTE: The Create File and Open File functions return an error code of 3 -> Path not Found or Invalid Pathname—as well as those error codes listed in the Microsoft and IBM literature.

There's one trick to the Rename File function, 56H. A file can be "moved" from one directory to another by including a different directory name in the second ASCIIZ string (the new pathname for the file).

Obtaining the file size is not straightforward. Function 42H must be used *after* opening the file, using AL = 2 -> move the file read/write pointer to the end of the file. The register pair DX:AZ indicates the size of the file in bytes (not logical records) on return from function 42H.

Although sequential and random I/O are possible with DOS 2 calls, these must be accomplished with fewer functions. Essentially, there is only one Read File function and one Write File function. Before you perform any I/O, the following steps must be performed:

1. If the file is to be used for sequential output, the file will typically be created with function 3CH. Otherwise, try opening your file using function 3DH with the appropriate value of AL:

   AL = 0 file is opened for read only
   AL = 1 file is opened for write only
   AL = 2 file is opened for reading and writing

2. If this open fails, use function 3CH to create a file.

Sequential I/O

Perform sequential I/O using DOS read function 3FH or write function 40H.

Random Record I/O

1. Knowing the record number to be read or written, calculate the byte offset of this record from the start of the file.

2. Use DOS function 42H, AL = 0, to move the file read/write pointer to this offset.

3. Set register CX equal to the logical record size.

4. Perform file I/O using DOS read function 3FH or write function 40H.

Random Block I/O

1. Knowing the FIRST record number to be read or written, calculate the byte offset of this record from the start of the file.

2. Use DOS function 42H, AL = 0, to move the file read/write pointer to this offset.

3. Set register CX equal to the *product* of the logical record size and the number of records to be read or written.

4. Perform file I/O using DOS read function 3FH or write function 40H.

NOTE: Descriptions of DOS functions 3FH and 40H in the Microsoft and IBM literature do not mention the effect of I/O on the file read/write pointer. The file I/O pointer is automatically moved forward during read/write operations. The distance moved is equal to the number of bytes actually read or written.

For this reason, sequential read and write operations can proceed without using the DOS function 42H -> move file read/write pointer.

Only during random I/O must the function 42H be used to posi-

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

tion the file read/write pointer before using function 3FH or 40H. Typically, function 42H is used with AL = 0 -> set the file read/write pointer to the designated offset from the start of the file.

Very rarely will function 42H be used with AL = 1 -> move pointer to current location plus offset.

Two More I/O Functions

Two commonly-needed file I/O functions not obviously accounted for are "adjust random access file size" and "flush buffer contents."

Adjust Random Access File Size

The only way to downward-adjust the size of the file is as follows:

1. Knowing the LAST record number to be retained in the file, calculate the byte offset of the byte beyond the end of this last record.
2. Use DOS function 42H, AL = 0, to move the file read/write pointer to this offset.
3. Set register CX equal to ZERO.
4. Use function 40H to set the size of the file. This special use of function 40H is mentioned only in the Microsoft *Programmer's Reference Manual*.

Flush Buffer Contents

This involves using DOS function 45H. The documentation is not clear as to the purpose of this function—perhaps it is included specifically for flushing file buffers!

1. After opening a file and performing I/O on it, call function 45H with the original file handle.
2. Function 45H returns a new handle in the register AX. Use this new handle with function 3EH to close the duplicate handle.
3. This has the effect of flushing the file buffer to disk while leaving the original handle available for further I/O.

Summary of DOS 2 File I/O Functions

Create File:	3CH	Rename File:	56H
Open File:	3DH	Get File Size:	42H, AL = 2
Close File:	3EH	Resize File:	Special use of 40H
Delete File:	41H	Flush Buffer:	Special use of 45H

Sequential I/O:		Read:	3FH	Write: 40H
Random Record I/O:	Set Ptr: 42H	Read:	3FH	Write: 40H
Random Block I/O:	Set Ptr: 42H	Read:	3FH	Write: 40H

□

Reference:

Microsoft MS-DOS Operating System Programmer's Reference Manual, Part No. 036-014-003, Document No. 8411-200-01.

Microsoft Corporation
Box 97017
10611 NE 36th Way
Redmond, WA 98073
(206) 882-8080

Sort RM/COBOL Files

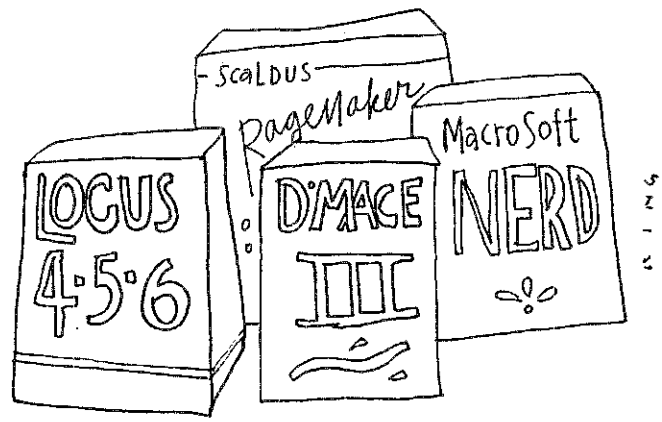
• • • QISORTC • • •

Finally, a sort utility for RM/COBOL.
Using efficient algorithms in assembly language making it the fastest sort on the MS-DOS market. Qisortc supports:

- RM/COBOL Version 1.5x & 2.xx files
  - Single / Dual File Indexed
  - Relative and Sequential
- ASCII, COMP-1, & COMP-3 fields
- 32 sort control fields
  - ascending / descending
- 32 record selection fields
  - EQ:NE:GT:GE:LT:LE field:constant
  - implied AND, OR may be specified
- 32 output fields or total record
- CALL from RM/COBOL, when provided subroutine is linked with runtime
- Use pathnames and return codes
- Keyboard or Parameter file
- Requires MS-DOS ver 2.xx or above
- Only $95.00 - COD, VISA, or M/C.

Quantumn Information Systems, Inc.
145 N.W. 85th St., Suite 103
Seattle, Washington 98117
(206) 789-2888 Telex 4941041

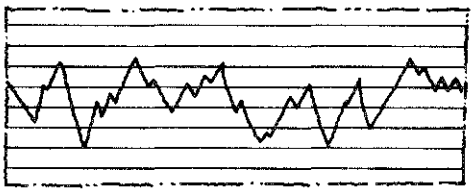
Circle Number 31 on the Reader Service Card



Pseudo applications Programs

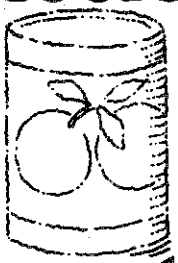
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

VEDIT® Plus Text Editor

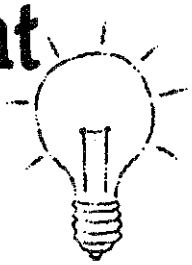


The Navy charts new concepts with it... GM engineers the future with

it...



National Can preserves facts with it...GE has bright ideas with it... Here's why you shouldn't be without it.



Every day, VEDIT PLUS helps thousands of programmers, writers and engineers get down to business.

So why do people who could have ANY text editor prefer ours? For a lot of reasons, including:

- CAPACITY—With VEDIT PLUS, file size is never a problem. And virtual disk buffering simplifies editing of even the largest files.
- FLEXIBILITY—VEDIT PLUS lets you edit up to 37 files simultaneously. So you can cut and paste. Edit programs. Edit text. Even perform numerous search/replace functions in several files without user intervention.*
- CUSTOMIZATION—With VEDIT PLUS, you can create your own on-line editing functions with keystroke macros. Develop your own on-line help screens. Determine and revise your own keyboard layout easily.
- SPEED—VEDIT PLUS not only works hard, it works fast. Faster, in fact, than any other text editor on the market.
- EXPERIENCE—Six years ago, CompuView revolutionized the concept of microcomputer text editing. And we've been improving our products and services ever since.

Special Offer: Order a VEDIT PLUS text editor for $225 and we'll include our V-PRINT™ document formatter—a $120 value—absolutely free.

Call CompuView today at 313/996-1299. You'll be in good company.

CompuView®

CompuView® Products Inc., 1955 Pauline Boulevard—Suite 300, Ann Arbor, Michigan 48103, TELEX 701821

Available for PC DOS, MS-DOS, CP/M, CP/M-86.

*Free sort, compare, print and main menu macros included; optional 8080-8086 translator or mailmerge, $50 each.

Circle Number **56** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Programming the WordStar Document File Conversion Utility: Again and Again and Again . . .

by Michael Hoyt

New Versions

As you gain programming experience, a review of previously written utility programs provides a rude awakening. Let's face it. What was once thought to be the "perfect" program no longer looks so exemplary. I've had this happen to me so often, I've come to realize that no program was or ever will be perfect—at least none I've written.

Actually, when first implemented, a program can be perfect for meeting a specific set of needs. But as those needs shift and/or change, the program loses its perfection. Not only does the target move, but new targets appear as old ones shift or disappear altogether. Moving and changing targets require new program versions.

I know almost any program I write will require new versions at some future date. Therefore, I use "good sense" and follow these four guidelines when programming.

1. Use lots of explanatory comments. Otherwise, when it's time for an update, it's difficult to decide what a program is doing. You may even have to throw it away and start over.
2. Write structured programs. With a structure, you can easily understand current program execution and follow the intended path. That makes it simpler to decide what changes are required.
3. Use "commented" subroutines to handle specific tasks. If each subroutine is well documented, replacing or rewriting is simplified; few problems will be caused by the rewrite.
4. Write the "perfect" program. That is, write the "best" program you can to fit current conditions. If the other three guide lines are followed, you have to worry about updates only when they're needed.

WordStar to ASCII

WS-ASCII.COM is an example of a program that has required rewriting—several times. The first writing of WS-ASCII was, alas, not even fit for revision. The sparse comments I'd used made it very difficult to follow the program logic. Using the term "structure" in reference to the original program is, well, generous. However, I must say I did the best I could, given my abilities at the time. Fortunately, experience has enabled me to improve on the original. I hope it will again

Since it was apparent that revising the original was a waste of time, I scrapped it. WS-ASCII, Version 2 became my first program truly designed for later revision. Over the years, that design decision has paid off.

The remainder of this article covers the sixth and latest revision performed on WS-ASCII. I choose to discuss this particular program for two reasons. First, a conversion program of this type is invariably one of the first "major" programs for most programmers. Many of you will have already written your version of WS-ASCII and therefore will be familiar with what's required. Second, the *PJ* editorial staff uses this conversion program (in this and other forms) on articles submitted for publication. The generic output is edited, then passed on to the typesetter. If you've any doubts that a "utility" program has use other than personal, here's the proof.

New Features

WS-ASCII, Version 5, had two features common to other WordStar conversion utilities. It not only stripped (zeroed) the 8th bit but removed formatting codes as well. Version 6 was written to meet additional requirements.

Most word processors currently available use their own "special" file format. But many of the same word processors can handle files in other formats. A specific format that can be accommodated by a growing number of word processors and editors is identical to the output from WS-ASCII. It consists of 7-bit ASCII characters, no special formatting codes, spaces rather than tab characters, carriage-return line-feed (CR-LF) at the end of paragraphs only (not at the end of each line), and no right-justification (no extra spaces between words).

Many word processors can output such a file format directly, including WordStar and MicroSoft Word. But some users either don't know how (or refuse) to save files in this format, so there's a distinct need for WS-ASCII.

Version 5 did not give the user the option of naming the output file, but defaulted to <filename>.NEW. When I was coding Version 6, it seemed worthwhile to allow assigning of an output filename. Still, it's a good idea to have a default name available if the user neglects to request one. So if an output filename is not requested, the default <filename>.NEW is used.

Since some users prefer to handle tasks with batch files, WS-ASCII was coded to return an "exit code" to DOS when it has finished execution. As DOS evolves into a more powerful operating system, the use of exit codes will become more prevalent, so it makes sense to include the feature now rather than later.

Normally, I would initially discuss the main procedure. But this time I have a specific reason to save it for later.

I've given each subroutine a heading. Typically the heading states what the subroutine does and lists the registers changed by its execution. If the flags are changed in any way, that information is also included. If, at some future date, it becomes necessary to rewrite (or replace) the routine, I can easily tell what must be done. In the long run, including a header for each subroutine is a time saver.

It's also a good idea to use descriptive labels. Many programmers still insist on abbreviating labels: a carry-over from the old days when compilers and assemblers allowed only 6–8 character labels. Now that being cryptic is no longer necessary, don't be. Trying to figure out what a label

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

means in an old program is more trouble than it's worth. As you look at WS-ASCII, I think you'll find that English labels make it easier to follow.

Syntax

WS-ASCII is a command-line utility. That is, one includes the parameters it needs on the command line. The syntax for use is: WS-ASCII <input filename.ext> <output filename.ext>. A drive letter and/or path name may be included as part of either (or both) input or output filenames.

Get_Input_Filename

To find the name of the input file, this routine uses information passed by DOS in the Program Segment Prefix (PSP). It contains the default Disk Transfer Address (DTA) at offset 80h. The first byte of the DTA indicates the number of characters contained on the command line. After that byte is checked for an ASCII zero, (it will be zero only if the command line was empty), the following characters (up to the first blank or carriage-return) are moved to a buffer for later use.

As in many of WS-ASCII's other routines, the carry flag is set to zero (0), which specifies that the routine was completed successfully. But if an error occurs, the carry flag is set to one (1), indicating to the main procedure that an error has occurred.

Get_Output_Filename

When this routine is called, the [AL] register still contains the last character read from the DTA. This saves having to fetch it, (not to mention program execution time). If more input is present in the DTA, it's assumed to be the name requested for the output file. But if all remaining input is found to be spaces followed by a carriage-return, the input filename is duplicated and assigned the extension NEW. This then becomes the name of the output file. Again, the error condition is passed using the carry flag.

Open_Input and Open_Output

These two (2) routines open the input and output files by using the DOS function calls. These particular DOS functions (3dh and 3ch) are used because they not only allow a file to be opened for "read only" or "write only," but in the case of DOS function call 3ch, allow a file to be created if it does not already exist. All this is done with one (1) call, thereby relieving you (the programmer) from having to provide the necessary code.

An additional benefit of using these specific DOS function calls is that you don't have to set the carry flag to indicate an error condition. DOS does it for you. You may have trouble finding this feature documented in the IBM DOS manual, (it's not there), but fortunately MicroSoft included the information in their version of the DOS reference manual.

Initialize_Buffers

After you open both input and output files, it's obvious that you need the text contained in the input file to be readily available for processing. In order to maximize the file processing speed, it makes sense to read a block of text from the input file into a buffer rather than one character at a time. That's just too slow.

This routine does nothing more than call another routine (Read_One_Block) that reads the specified number of characters into Input_Buffer, then sets Output_Pointer to the beginning of Output_Buffer.

Read_One_Block

This is the routine used to read character blocks into the input buffer for processing. Notice that the number of characters read is determined by the EQUate, Block_Size. The size of both buffers (Input_Buffer and Output_Buffer) is also determined by Block_Size. This allows me (or you) to easily change the amount of memory required for using WS-ASCII. The current Block_Size (1024 bytes) seems about right for operation on most systems, even ones with a fairly small amount of memory.

Like the other routines used in WS-ASCII, the READ routine used a DOS function call rather than reading the diskette file directly or using ROM_BIOS routines. Unless DOS changes drastically in the future, this sort of programming eliminates re-writes to stay compatible. This particular DOS function (3f) also sets the carry flag to indicate an error condition, which is in keeping with the error handling pattern used in other portions of the program.

Main Procedure

Normally I structure a program with a main procedure that calls other modules (subroutines), each performing one function. Limiting each subroutine to one basic function makes it simpler if the module requires later rewriting. However, my programs tend to include two functions as part of the main procedure. The first is a check for the DOS version. Second is the handling of program exit, including error conditions severe enough to cause the program to stop execution. Although many prefer to do both in subroutines, it's simpler for me if each is part of the main procedure.

If you look closely at WS-ASCII, you'll notice that the routine that tests each individual character (Get_Character) is performed as part of the main procedure rather than as a subroutine. The reason for this seeming deviation from my standard program design is simple. Speed. A call to a subroutine requires 35 clock cycles (CALL-19, RET-16). Since each input character is handled one at a time, an addition of 35 cycles for each adds a significant length of time to program execution. Testing each character already requires the use of multiple Jxx (jump) instructions, each capable of requiring 16 clock cycles (if jump taken). So it makes sense to eliminate a CALL-RET for each character.

Actually, the portion of code from Get_Character up to (but not including) Close_Files is a subroutine. It's just placed so no CALL (and subsequent RET) is needed for its execution. In fact, even more time can be saved by moving another routine, Get_A_Character, to the beginning of this code section, (labeled Get_Character), thereby eliminating another CALL-RET combination. I'll leave that improvement for you to do.

Get_A_Character

Now that the files (input and output) are opened and the first block of characters has been read into the buffer for processing, this procedure is used to fetch each character for processing. When the last character in the buffer is reached, the routine also takes care of calling the previously discussed routine (Read_One_Block) for reading another block of characters from the input file.

Get_Character

Although there are other ways (some better) to check each input character for ones needing special handling, I've chosen to use a set of compare (CMP) instructions. The logic is fairly straightforward and should be easy for you to follow. If a specific character is found to need special treatment, execution jumps to the needed code, such as Soft_Return, Expand_Tab_Character, . . .

After each character is processed, the Put_Character routine is called.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Periscope Is On Sale!

"A marvelous debugging tool..." says Peter Norton.

SUMMER SALE. Through August 31, the new software-only Periscope II-X is yours for just $95! Periscope II, normally $145, is only $125!

NEW MODEL. Periscope I includes both a break-out switch for "spontaneous" debugging and a RAM board to protect the debugger from programs that overwrite memory. Periscope II includes a break-out switch that doesn't require an extra slot to install. *And the new Periscope II-X is Periscope II without the break-out switch. So you can now get Periscope with no hardware at all!*

NEW RELEASE. The new version 2.1 enhances all models with many of the features users have requested. Call for details.

CHECK OUT THE REVIEWS.

"Periscope strikes a balance of power and features that will please serious PC programmers." —Jeff Duntemann, PRODUCT OF THE MONTH, *PC Tech Journal*, 1/86

"Periscope's diverse features, affordable price, and portability place it in a class by itself." —Ward Christensen, "Breaking out with Periscope," *PC Tech Journal*, 3/86

"Periscope represents the finest software debugger available in its class." —Andrew Fried, *Computer Shopper*, 4/86

REQUIREMENTS. An IBM PC, XT, AT or close compatible; 128K RAM; DOS 2.0 or later; one disk drive; and an 80-column monitor.

Periscope II-X, Software Only ..... $ 95*
Periscope II, with Switch $125*
Periscope I, with Board & Switch .. $295
*Prices good through August 31, 1986

Take advantage of the low sale prices while they last . . . Order your Periscope today! Call toll-free:

800/722-7006

30-Day Money-Back Guarantee

Shipping-$3 UPS ground; $5 COD; $8 Canada; $24 Foreign. We accept qualified PO's in the U.S.

Get Your Programs Up and Running;

UP PERISCOPE!

Data Base Decisions • (404) 256-3860
14 Bonnie Lane • Atlanta, GA 30328

Circle Number 6 on the Reader Service Card

Put_Character

Put_Character is used to place each processed character into Output_Buffer and if necessary call the routine Write_One_Block to write the full buffer to disk. Notice that each of these routines also uses the carry flag to indicate error conditions.

Close_Files

Assuming everything has gone according to plan, this section of code is reached after the input file has been completely processed. Its first obligation is to place the end-of-file (EOF) marker after the last character placed in Output_Buffer. This ensures that even if the input file had no EOF, the new file will. Next, the last (and probably partial) block of text in the output buffer must be written to disk. A call to Flush_Buffers takes care of that.

Even though many wouldn't bother to close the input file, I always do. Only so many files may be opened at one time under the current versions of DOS. If files are left open, problems will certainly occur. And of course the output file must be closed in order to force DOS to write its file buffer to disk.

Ending Up

When I write a utility such as this, I make the display of an "ending" message part of the routine. Otherwise the user of the utility may be at a loss as to when (or if) the utility is finished.

As mentioned before, I also use the DOS function 4C for passing an error (or Return_Code, as I call it) to DOS. Since MicroSoft may eventually decide upon a standard set of codes for this purpose, I've made Return_Code an EQUate so that it can easily be changed.

Challenge

There are several places in WS-ASCII where improvements can be made—especially speed improvements. One example is to make the subroutine Get_A_Character part of the Main routine by insertion at Get_Character. At least one other subroutine also could be made a part of Main. It's possible (and probable) that certain sections of code can be optimized for speed.

The challenge is this. For the next several months, I'll take suggestions for improving WS-ASCII. Suggestions sent on disk get preferential treatment. Make sure your suggestions include well commented source code and please, debug the changes before you send them. After looking at the suggestions (subjectively, of course), we'll print the new WS-ASCII in an upcoming issue of *PJ*. Of course the new version will include the names and code of those who contributed the improvements.

In the interim, if you want a complete listing and working copy of this version (for non-commercial use only), I'll be happy to send them to you on a new diskette for a $15.00 handling fee. □

Michael Hoyt is a contract programming consultant and freelance writer who frequently contributes to Programmer's Journal. *Readers may write to him at*

RR 2, Box 65
Solsberry, IN 47459

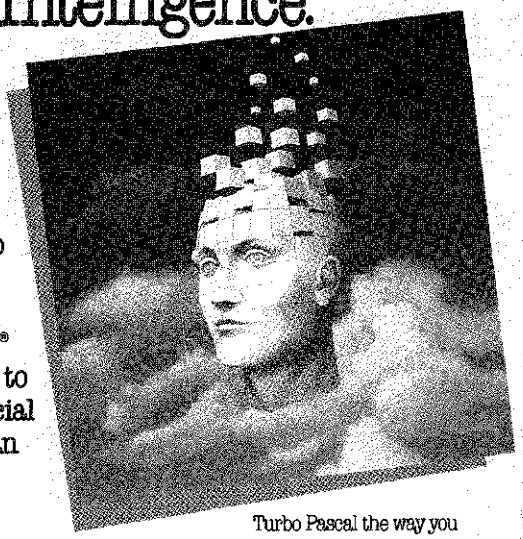
Code follows.

Jocelyn realizes the new DOS manual was in Chinese... and much easier to read than the old copy.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Step-by-step tutorial, demo programs with source code included!

Borland introduces Turbo Prolog, the natural language of Artificial Intelligence.

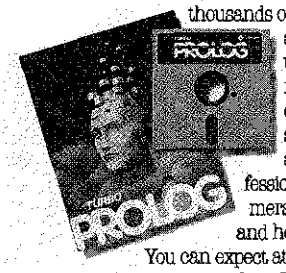


Prolog is probably the most powerful computer programming language ever conceived, which is why we've made it our *second* language—and "turbocharged" it to create Turbo Prolog.™

Our new Turbo Prolog brings supercomputer power to your IBM® PC and introduces you step-by-step to the fascinating new world of **A**rtificial **I**ntelligence. And does all this for an astounding $99.95.

Turbo Prolog is to Prolog what Turbo Pascal® is to Pascal!

Our Turbo Pascal astonished everyone who thought of Pascal as "just another language." We changed all that—and now Turbo Pascal is the de facto worldwide standard, with hundreds of thousands of enthusiasts and users in universities, research centers, schools, and with professional programmers, students, and hobbyists.



You can expect at least the same impact from Turbo Prolog, because while Turbo Prolog is the most revolutionary and natural programming language, it is also a complete development environment—just like Turbo Pascal.

Turbo Prolog radically alters and dramatically improves the brave new world of artificial intelligence—and invites you into that fascinating universe for a humanly intelligent $99.95.

Even if you've never programmed before, our free tutorial will get you started right away

You'll get started right away because we have included a complete step-by-step tutorial as part of the 200-page Turbo Prolog Reference Manual. Our tutorial will take you by the hand and teach you everything you're likely to need to know about Turbo Prolog and artificial intelligence.

For example: once you've completed the tutorial, you'll be able to design your own expert systems utilizing Turbo Prolog's powerful problem-solving capabilities.

Think of Turbo Prolog as a high-speed electronic detective. First you feed it information and teach it rules. Then Turbo Prolog "thinks" the problem through and comes up with all the reasonable answers—almost instantly.

If you think that this is amazing, you just need to remember that Turbo Prolog is a 5th-generation language—and the kind of language that 21st century computers will use routinely. In fact, you can compare Turbo Prolog to Turbo Pascal the way you could compare Turbo Pascal to machine language.

You get the complete Turbo Prolog programming system for only $99.95

You get a complete Turbo Prolog development system including:

- The lightning-fast Turbo Prolog incremental compiler and the interactive Turbo Prolog editor.
- The 200-page reference manual which includes the step-by-step Turbo Prolog tutorial.
- The free GeoBase™ natural query language database including commented source code on disk—ready to compile. GeoBase is a complete database designed and developed around U.S. geography. It includes cities, mountains, rivers, and highways, and comes complete with natural query language. Use GeoBase immediately "as is," or modify it to fit your own interests.

So don't delay—don't waste a second—get Turbo Prolog now. $99.95 is an amazingly small price to pay to become an immediate authority—an instant expert on artificial intelligence! The 21st century is only one phone call away.

Turbo Prolog 1.0 Technical Specifications

Programming System Features

☑ *Compiler:* Incremental compiler generating native in-line code and linkable object modules. The linking format is compatible with the PC-DOS linker. Large memory model support. Compiles over 2500 lines per minute on a standard IBM PC.

☑ *Interactive Editor:* The system includes a powerful interactive full-screen text editor. If the compiler detects an error, the editor automatically positions the cursor appropriately in the source code. At run-time, Turbo Prolog programs can call the editor, and view the running program's source code.

☑ *Type System:* A flexible object-oriented type system is supported.

☑ *Windowing Support:* The system supports both graphic and text windows.

☑ *Input/Output:* Full I/O facilities, including formatted I/O, streams, and random access files.

☑ *Numeric Ranges:* Integers: −32767 to 32767; Reals: 1E−307 to 1E+308

☑ *Debugging:* Complete built-in trace debugging capabilities allowing single stepping of programs.

YES! I want the best

Turbo Prolog at only:

$99.95

To order by phone, or for a dealer nearest you,

Call (800) 255-8008

in CA call (800) 742-1133.

Send me ____ Turbo Prolog at $ ____

Outside USA add $10 per copy
CA and MA res. add applicable sales tax $ ____

Amount enclosed: $ ____

This price includes shipping to all US cities

Payment: VISA MC Bank Draft Check

Credit card expiration date: ___/___

Card #

*You must have an IBM or true compatible running DOS 2.0 or later.***
My computer's name and model is:

The disk size I use is: ☐ 3½" ☐ 5¼"

NOT COPY PROTECTED
***60-DAY MONEY-BACK GUARANTEE**

Name:

Shipping Address:

City:

State: *Zip:*

Telephone:

CODs and purchase orders WILL NOT be accepted by Borland. Outside USA make payment by credit card or International Postal Money Order.

**YES, if within 60 days of purchase this product does not perform in accordance with our claims, please call our customer service department and we will gladly arrange a refund.*

***** Minimum system requirements:***
IBM PC, XT, AT, PCjr, and true compatibles
384K RAM

SU11

BORLAND INTERNATIONAL
Vive la différence

4585 SCOTTS VALLEY DRIVE
SCOTTS VALLEY, CA 95066
(408) 438-8400 TELEX: 172373

Other Borland Products include Turbo Pascal; Turbo Tutor; Turbo Lightning; Turbo Database Toolbox; Turbo Graphix Toolbox; Turbo Editor Toolbox; Turbo GameWorks; SuperKey; SideKick; SideKick, The Macintosh Office Manager; Reflex, The Analyst; and Traveling SideKick—all of which are registered trademarks or trademarks of Borland International, Inc. or Borland/Analytica, Inc.
Turbo Prolog and GeoBase are trademarks and Turbo Pascal is a registered trademark of Borland International Inc. IBM and AT are registered trademarks of International Business Machines Corp. Copyright 1986 Borland International BI-1045D

Circle Number 3 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

1

```
page    60,132
name    ws_ascii
title   WordStar to ASCII conversion utility

;:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;                                                                               ;
;                               WordStar to ASCII                               ;
;                                   Version 6                                   ;
;           (c) Copyright 1983, 1984, 1985, 1986 by Michael L Hoyt              ;
;                               All rights reserved.                            ;
;                                                                               ;
;  This procedure is used to convert WordStar Document files to an ASCII        ;
;  format which can then be used by a number of text editors and word           ;
;  processors.  The newly created file is named <filename>.NEW.                 ;
;                                                                               ;
;  The program performs the following operations on the original file.          ;
;                                                                               ;
;       1) Strips the high-bit - all output is in 7-bit ASCII                   ;
;       2) Removes all formatting codes.                                        ;
;       3) Removes all "soft" spaces - removes "right-justification".           ;
;       4) Removes all "soft carriage return line feed combinations -           ;
;          CR-LF combinations remains at end-of-paragrah only.                  ;
;                                                                               ;
;  Procedure returns error codes to DOS upon exit.                              ;
;       00h = procedure completed ok                                            ;
;       ffh = procedure failed to complete                                      ;
;                                                                               ;
;  For use with DOS 2.0 or later.                                               ;
;                                                                               ;
;                                                                  [11 feb 86]  ;
;:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

        .radix  16
Code    segment byte
        assume  cs:Code,ds:Code,es:Code ;no stack segment for COM files
        org     100                     ;starting point for COM files

;================================< Equates >=====================================

Cr              equ 0dh                 ;carriage return
Lf              equ 0a                  ;line feed
Ff              equ 0c                  ;form feed
Eof             equ 1a                  ;end of file marker
Tab             equ 09                  ;tab character
Dta             equ 0080                ;data transfer offset in PSP
Block_Size      equ 0400                ;size of block read from file

page ;=========================< Main Procedure >=================================

WS_Ascii        proc far
        mov     ah,30                   ;get the version of DOS being used
        int     21                      ;use DOS function call
        cmp     al,02                   ;is it 2.0 or later ?
        jae     Process_Input           ;yes, so process the command line input
        mov     dx,offset Message_6     ;no, point to correct error message
        mov     byte ptr Return_Code,0ff        ;set exit code to show error
        jmp     Display_Message         ;display it and exit
Process_Input:
        call    Get_Input_Filename      ;get the user specified filename, etc.
        jnc     Filename_Ok             ;the filename is ok, continue
        mov     dx,offset Message_4     ;point to error message
        mov     byte ptr Return_Code,0ff        ;set exit code to show error
        jmp     Display_Message         ;display error message
Filename_Ok:
        call    Get_Output_Filename     ;name the output file
        call    Open_Input              ;open the input file
        jnc     Input_File_Opened       ;opened ok, so continue
        mov     dx,offset Message_1     ;open failed, point to error message
        mov     byte ptr Return_Code,0ff        ;set exit code to show error
        jmp     Display_Message         ;display error message
Input_File_Opened:
        call    Open_Output             ;open the new file for output
        jnc     Output_File_Opened      ;opened ok, so continue
        mov     dx,offset Message_2     ;open failed, point to error message
        mov     byte ptr Return_Code,0ff        ;set exit code to show error
```

2

```
        jmp     Display_Message         ;display error message
Output_File_Opened:
        call    Initialize_Buffers      ;get 1st block into buffer
Get_Character:
        call    Get_A_Character         ;read a character from the input file
        cmp     al,0a0                  ;is this a 'soft' space ?
        je      Get_Character           ;yes, so discard and get next character
        cmp     al,8dh                  ;is it a 'soft' Carriage Return ?
        je      Soft_Return             ;yes, handle a special way
        and     al,7f                   ;strip off the high bit
        cmp     al,20                   ;is it a control code ?
        jae     Update_Column_Count     ;no, so add to new file
        cmp     al,Eof                  ;is it the EOF ?
        je      Close_Files             ;yes, so close files
        cmp     al,Tab                  ;is it a tab ?
        je      Expand_Tab_Character    ;yes, so expand tab
        cmp     al,Cr                   ;is it a carriage return ?
        je      Zero_Column_Count       ;yes, so reset column to (0)
        cmp     al,Ff                   ;is it a form feed ?
        je      Zero_Column_Count       ;yes, so reset column to (0)
        cmp     al,Lf                   ;is it a line feed ?
        jne     Get_Character           ;if not, throw it away
                                        ; and get the next character
        cmp     byte ptr Return_Flag,01 ;does this LF follow a 'soft' return ?
        jne     Zero_Column_Count       ;no, so handle standard way
        xor     byte ptr Return_Flag,01 ;yes, clear the flag, throw away this
        jmp     Get_Character           ;character and get the next one
page
Soft_Return:
        xor     byte ptr Return_Flag,01 ;set the 'soft' indicator
        mov     word ptr Column,0000    ;set column count back to zero
        jmp     Get_Character           ;and get the next character
Zero_Column_Count:
        mov     word ptr Column,0000    ;set column count back to zero
        jmp     short Write_A_Character ;put the character in the new file
Update_Column_Count:
        inc     word ptr Column         ;update character count
Write_A_Character:
        call    Put_A_Character         ;write this character to the file
        jnc     Get_Character           ;continue if disk is not full
        mov     dx,offset Message_5     ;point to error message
        mov     byte ptr Return_Code,0ff        ;set exit code to show error
        jmp     short Disk_Full_Error   ;close files and display message
Expand_Tab_Character:
        mov     ax,Column               ;get column count into [AX]
        cwd                             ;prepare for double word operation
        mov     cx,0008                 ;use [CX] as the divisor
        idiv    cx                      ;divide by 8
        sub     cx,dx                   ;subtract remainder from 8
        add     word ptr Column,cx      ;update column count
Space:
        mov     bp,cx                   ;save the space count
        mov     al,20                   ;put the space character in [AL]
        call    Put_A_Character         ;place it in the new file
        mov     cx,bp                   ;recover the space count
        loop    Space                   ;loop until done
        jmp     Get_Character           ;get the next character
Close_Files:
        call    Put_A_Character         ;write the EOF to the new file
        call    Flush_Buffers           ;send last info to new file
        mov     dx,offset Message_3     ;point to standard exit message
Disk_Full_Error:
        mov     bx,Input_Handle         ;point to input file handle
        call    Close_File              ;close the input file
        mov     bx,Output_Handle        ;point to output file handle
        call    Close_File              ;close the output file
Display_Message:
        mov     ah,09                   ;set [AH] to display string
        int     21                      ;display string pointed to by [DX]
        mov     ah,4c                   ;use DOS function to pass
                                        ; return code contained in [AL]
        mov     al,Return_Code          ;recover return code
        int     21                      ;exit to DOS
WS_Ascii        endp
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
page :=========================< Subroutines >================================

;-----------------------------------------------------------------------------
; Get the filename from the command-line input.
;
; CHANGED - [AX], [SI], [DI], flags
;-----------------------------------------------------------------------------
Get_Input_Filename      proc near
        cld                             ;set the direction flag to forward
        mov     si,offset Dta           ;point to the command line string
        mov     di,offset Old_Filename  ;point to the filename buffer
        lodsb                           ;see if a command line is present
        or      al,al                   ;is there a size indicator ?
        jz      Input_Error             ;no, indicate error
Scan_Input:
        lodsb                           ;search for first non-blank character
        cmp     al,Cr                   ;have we reached a carriage return ?
        je      Input_Error             ;yes, so there is no command line
        cmp     al,20                   ;is this a blank ?
        je      Scan_Input              ;yes, so get next input character
Store_Input:
        stosb                           ;no, this is a character, save it
        lodsb                           ;check the next character
        cmp     al,Cr                   ;is it a carriage return ?
        je      Input_Ok                ;yes, so indicate input is alright
        cmp     al,20                   ;is it a space ?
        jne     Store_Input             ;no, so keep moving characters
Input_Ok:
        clc                             ;set carry flag to = 0
        ret                             ;done, so return
Input_Error:
        stc                             ;set carry flag to = 1
        ret                             ;done, so return
Get_Input_Filename      endp

page :-----------------------------------------------------------------------
; Set the output filename.  Recover from the command-line input if present.
;
; CHANGED - [AX], [SI], [DI]  flags
;-----------------------------------------------------------------------------
Get_Output_Filename     proc near
        mov     di,offset New_Filename  ;point to the destination file
        cmp     al,Cr                   ;did we read the last input character ?
        je      Duplicate_Filename      ;yes, so use the input filname
                                        ; with a new extension
        cmp     al,20                   ;is this character a space ?
        jne     Store_Output_Filename   ;no, so save it
        jmp     short Discard_Space     ;yes, so discard it
Store_Output_Filename:
        stosb                           ;this is a character, save it
Discard_Space:
        lodsb                           ;check the next character
        cmp     al,Cr                   ;is it a carriage return ?
        je      Name_Ok                 ;yes, so return
        cmp     al,20                   ;is it a space ?
        jne     Store_Output_Filename   ;no, so keep moving characters
Name_Ok:
        ret                             ;done, so return
Duplicate_Filename:
        mov     cx,0040                 ;set [CX] for length of move
        mov     si,offset Old_Filename  ;point to name of the source file
        rep     movsb                   ;move a byte-at-a-time until done
        mov     di,offset New_Filename  ;point to start of output filname
New_Extension:
        mov     cx,0040                 ;place count in [CX]
        xor     al,al                   ;put an ASCII zero in [AL]
        repnz   scasb                   ;find the end of the filename
        dec     di                      ;set pointer to end of ASCIIZ string
        mov     si,di                   ;save the value for later
        std                             ;reverse the scan direction
        mov     al,2e                   ;place the period character in [AL]
        mov     cx,0006                 ;place the count in [CX]
        repnz   scasb                   ;see if a filename extension
                                        ; is included
```

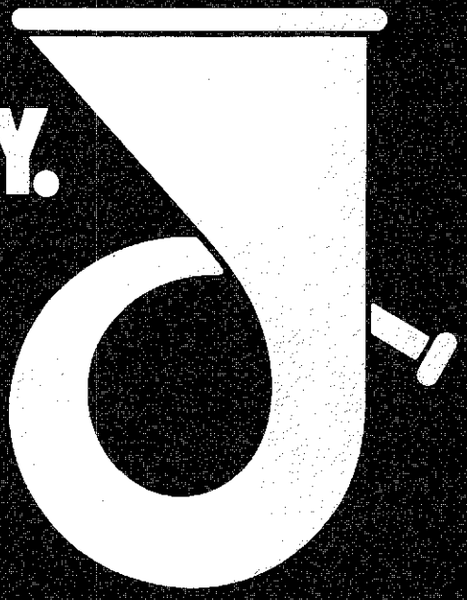


```
        jcxz    Add_Extension           ;no extension included, so add one
        inc     di                      ;extension included, point to period
        cld                             ;set direction forward
        jmp     short Set_Extension     ;change extension to "NEW"
Add_Extension:
        mov     di,si                   ;recover pointer to end of filename
Set_Extension:
        mov     si,offset Extension     ;point to the new filename extension
        mov     cx,0004                 ;set number of bytes to be moved
        rep     movsb                   ;move a byte at-a-time until done
        ret                             ;done, so return
Get_Output_Filename     endp

page :-----------------------------------------------------------------------
; Open the user requested file for input.
;
; CHANGED - [[AX], [DX], flags
;-----------------------------------------------------------------------------
Open_Input      proc near
        mov     dx,offset Old_Filename  ;point to input filename
        mov     ax,3d00                 ;[AL] = read only  |  [AH] = open file
        int     21                      ;use DOS function call
        mov     Input_Handle,ax         ;save handle for later
        ret                             ;done, so return
Open_Input      endp

;-----------------------------------------------------------------------------
; Create and open the output file.
;
; CHANGED - [AX], [DX], flags
;-----------------------------------------------------------------------------
Open_Output     proc near
        mov     dx,offset New_Filename  ;point to output filename
        mov     ax,3c01                 ;[AL] = write only, [AH] = make file
        int     21                      ;DOS function call
        mov     Output_Handle,ax        ;save handle for later
        ret                             ;done, so return
Open_Output     endp

;-----------------------------------------------------------------------------
;  Close the requested file handle.
;
; INPUT   - [BX] = handle of file to be closed
;
; CHANGED - [AX], flags
;-----------------------------------------------------------------------------
Close_File      proc near
        mov     ah,3e                   ;DOS function to close file
        int     21
        ret                             ;done, so return
Close_File      endp

page :-----------------------------------------------------------------------
; Read the 1st block into the input buffer and set the pointer for the
; output buffer.
;
; CHANGED - [AX], [BX], [CX], [DX], carry flag
;-----------------------------------------------------------------------------
Initialize_Buffers      proc near
        call    Read_One_Block          ;read the 1st block of input
        mov     Output_Pointer,ax       ;point to beginning of output buffer
        ret                             ;done, so return
Initialize_Buffers      endp

;-----------------------------------------------------------------------------
; Make sure any remaining contents in output buffer is written to the
; output file.
;
; CHANGED - [AX], [BX], [CX], [DX], carry flag
```

(Continued...)

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

YOUR COMPUTER LANGUAGE IS QUIETLY BREEDING REAL BATS IN YOUR BELFRY.



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

WE'RE OUT TO SAVE ONE MILLION FRUSTRATED PROGRAMMERS

You're on a roll, really pumped, writing the best code you have ever written and then – AAARGHHH!

Freeze dried in your tracks because the language you're using just won't let you achieve what you can conceive.

And you wanted to be a programmer.

So your choices are:

1) write around the problem by creating six pages of emetic code...

2) leave out that incredible idea that really puts your stamp of excellence on this program or...

3) get yourself a world class headache (or a stroke) by dropping into assembler.

Whatever you choose, by now you feel the language is out to get you – because it is.

Sure, no language is perfect, but you have to wonder, "Am I getting all I deserve?"

And, like money, you'll never have enough.

Pretty dismal, huh?

We thought so, too.

So we did something about it.

We call it CLARION.

You'll call it incredible.

With CLARION you can write, compile, run and debug complex applications in a New York afternoon.

Even if you're in Savannah.

It gives you the power and speed to create screens, windows and reports of such richness and clarity you would never attempt them with any other language.

Because YOU would have to write the code.

With CLARION you simply design the screens using our SCREENER utility and then CLARION writes the source code AND compiles it for you.

In seconds.

Likewise, you can use REPORTER to create reports. Remember, only CLARION can recompile and display a screen or report layout for modification.

And with no time wasted.

All the power and facilities you need to write great programs, faster than you ever dreamed of.

Programs that are easy to use.

Programs that are a pleasure to write.

And to you that means true satisfaction.

You've coveted those nifty pop-up help windows some major applications feature. But you can't afford the time and energy it takes to write them into your programs.

That's the way it used to be.

So we fixed that, too.

CLARION HELPER is an interactive utility that let's you design the most effective pop-up help screens that you can imagine. And they're "context sensitive," meaning you can have help for every field in your application.

Unlike the other micro languages, CLARION provides declarations, procedures and functions to process dates, strings, screens, reports, indexed files, DOS files and memory tables.

Imagine making source program changes with the CLARION EDITOR. A single keystroke terminates the EDITOR, loads the COMPILER, compiles the program, loads the PROCESSOR and executes the program. It's that easy!

Our data management capabilities are phenomenal. CLARION files permit any number of composite keys which are updated dynamically.

A file may have as many keys as it needs. Each key may be composed of any fields in any order. And key files are updated whenever the value of the key changes.

Like SCREENER and REPORTER, CLARION's FILER utility also has a piece of the CLARION COMPILER. To create a new file, you name the Source Module. Then you name the Statement Label of a file structure within it.

FILER will also automatically rebuild existing files to match a changed file structure. It creates a new record for every existing record, copying the existing fields and initializing new ones.

Sounds pretty complicated, huh?

Not with CLARION's documentation and on-line help screens. If you are currently competent in Basic, Pascal or "C" you can be writing CLARION applications in a day. In two days you won't believe the eloquence of your CLARION programs.

Okay, now for the best part of all. You can say it in CLARION for $295.00 – complete. All you need is an IBM® PC, XT, AT or true compatible, with 320 KB of memory and a hard disk drive.

And we'll allow a full 30 day evaluation period. If you're not satisfied with CLARION, simply return it in its original condition for a full refund.

If you're not ready to take advantage of this no-risk opportunity, ask for our detailed 16 page color brochure. It vividly illustrates the elegance of CLARION. Consider it a preview of programming in the fast lane.

Either way, the call's a freebie.

SAY IT IN CLARION™

DEPT. AIPL/1

1-800-354-5444

VISA MasterCard

BARRINGTON SYSTEMS, INC. • P.O. BOX 5580 • POMPANO BEACH, FL 33074 • 305/785-4555

IBM® is a registered trademark of International Business Machines Corporation. CLARION™ is a trademark of Barrington Systems, Inc. ©1986 Barrington Systems.

Circle Number 55 on the Reader Service Card.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

5

```
;-----------------------------------------------------------------------------
Flush_Buffers   proc near
        mov     cx,Output_Pointer       ;get output pointer into [CX]
        or      cx,cx                   ;is there anything in buffer ?
        jz      Empty                   ;empty, but set flag
        mov     bx,Output_Handle        ;get the output file handle
        mov     dx,offset Output_Buffer ;point to the output buffer
        mov     ah,40                   ;DOS function to write file
        int     21
        cmp     ax,Output_Pointer       ;was the write successful ?
        jnz     Go_Back                 ;disk full, indicate error
Empty:
        clc                             ;indicate successful write
        ret                             ;done, so return
Go_Back:
        stc                             ;indicate write error
        ret                             ;done, so return
Flush_Buffers   endp

page ;-----------------------------------------------------------------------------
; Get one character from the input buffer and process it.
;
; CHANGED - [AX], [BX], [CX], [DX], carry flag
;-----------------------------------------------------------------------------
Get_A_Character proc near
        mov     bx,Input_Pointer        ;get the input buffer pointer
        cmp     bx,Block_Size           ;are we finished with this block yet
        jne     Get_One_Character       ;no, so get a character
        call    Read_One_Block          ;read the next block into the buffer
        xor     bx,bx                   ;point to 1st character in buffer
Get_One_Character:
        mov     si,offset Input_Buffer  ;point to beginning of input buffer
        mov     al,[bx+si]              ;get a character from the buffer
        inc     bx                      ;update [BX] to point to next character
        mov     Input_Pointer,bx        ;store the input pointer
        ret                             ;done, so return
Get_A_Character endp

;-----------------------------------------------------------------------------
; Read a block from the input file into the input buffer.
;
; CHANGED - [AX], [BX], [CX], [DX], carry flag
;-----------------------------------------------------------------------------
Read_One_Block  proc near
        mov     bx,Input_Handle         ;point to the input file handle
        mov     cx,Block_Size           ;set the # of characters to read
        mov     dx,offset Input_Buffer  ;point to the input file buffer
        mov     ah,3f                   ;DOS function to read file
        int     21
        jnc     Read_Error              ;jump if no read error
        xor     ax,ax                   ;simulate a zero read on error
Read_Error:
        cmp     ax,Block_Size           ;did we read in a full buffer ?
        je      Return                  ;yes, so return
        mov     bx,ax                   ;get input buffer pointer into [BX]
        add     bx,offset Input_Buffer  ;point to offset in the input buffer
        mov     byte ptr [bx],EOF       ;place EOF in input buffer
Return:
        xor     ax,ax                   ;set pointer to beginning of buffer
        mov     si,ax                   ;set pointer into the input buffer
        ret                             ;done, so return
Read_One_Block  endp

page ;-----------------------------------------------------------------------------
; Put a character into the output buffer.
;
; CHANGED - [AX], [BX], [CX], [DX], [DI], carry flag
;-----------------------------------------------------------------------------
Put_A_Character proc near
        mov     bx,Output_Pointer       ;point to the output buffer
        mov     di,offset Output_Buffer ;point to beginning of output buffer
        mov     byte ptr [bx+di],al     ;put the character in the buffer
        inc     bx                      ;update [BX] to point to next location
        mov     Output_Pointer,bx       ;store the output pointer
```

6

```
        cmp     bx,Block_Size           ;is the buffer full ?
        jne     Not_Full                ;no, so return
        call    Write_One_Block         ;yes, so write this block to the file
        ret                             ;done, so return
Not_Full:
        clc                             ;set flag for status report
        ret                             ;done, so return
Put_A_Character endp

;-----------------------------------------------------------------------------
; Write the output buffer to the output file.
;
; CHANGED - [AX], [BX], [CX], [DX], carry flag
;-----------------------------------------------------------------------------
Write_One_Block proc near
        mov     bx,Output_Handle        ;get the output file handle
        mov     cx,Block_Size           ;set [CX] to write 1 block
        mov     dx,offset Output_Buffer ;point to output buffer
        mov     ah,40                   ;DOS function to write to file
        int     21
        xor     bx,bx                   ;set pointer to beginning of buffer
        mov     Output_Pointer,bx       ;store the output pointer
        cmp     ax,Block_Size           ;see if the block was written OK
        jne     Write_Error             ;jump if the disk is full
        clc                             ;indicate successful write
        ret                             ;done, so return
Write_Error:
        stc                             ;indicate disk full - write not OK
        ret                             ;done, so return
Write_One_Block endp

page ;-----------------------------< Data >-----------------------------------

Message_1       db Cr,Lf
                db 'Input file not found ...'
                db Cr,Lf,24
Message_2       db Cr,Lf
                db 'Could not open output file ...'
                db Cr,Lf,24
Message_3       db Cr,Lf
                db 'Processing completed ...'
                db Cr,Lf,24
Message_4       db Cr,Lf
                db 'You must name the file to be processed ...'
                db Cr,Lf,24
Message_5       db Cr,Lf
                db 'Disk Full.  Processing not completed ...'
                db Cr,Lf,24
Message_6       db Cr,Lf
                db 'This procedure only works with DOS 2.0 or newer ...'
                db Cr,Lf,24
page
Old_Filename    db 40 dup (00)          ;storage for the input filename
New_Filename    db 40 dup (00)          ;storage for the output filename
Column          dw 0000                 ;storage for column count
Return_Flag     db 00                   ;soft carriage return flag
Return_Code     db 00                   ;storage for return error code
Extension       db '.NEW'               ;extension of new file
Input_Handle    dw ?                    ;token returned by DOS
Output_Handle   dw ?                    ;token returned by DOS
Input_Pointer   dw ?                    ;store pointer into input buffer
Output_Pointer  dw ?                    ;store pointer into output buffer
Input_Buffer    db Block_Size dup (?)   ;characters read from input file
Output_Buffer   db Block_Size dup (?)   ;characters to write to output file

Code    ends
End     WS_Ascii
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

If BASIC's data limit is the question . . .

DataSpace

. . . is the answer

DataSpace gives MS* and GW-BASIC* users . . .

- the ability to define and access string, integer, single, and double precision data arrays
- 10 - one, two or three-dimensional arrays up to 64K each - 640K total
- array storage and retrieval from files in any disc directory
- full compatibility with interpreters or compilers

. . . all with absolutely no impact on BASIC's own data space (requires MS-DOS* 2.10 or higher)

The only answer around for $75 . . .

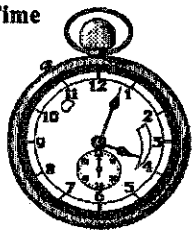
DESIGN SOLUTIONS INC.

732 Sixth Avenue
Aurora, IL 60505
(312) 898-0651

* MS, GW-BASIC, and MS-DOS are trademarks of Microsoft Corporation

Circle Number 41 on the Reader Service Card

Along With Your Computer, Your Time is the Most Important Thing You Own. . .So Why Waste It?



Quilt Programmer Productivity Tools will help you manage your software projects and get control of your time!

The Software Revision Management System (SRMS)™ stores all versions of source code/documentation in a single ASCII file. Allows retrieval of any version of source and application of changes while recording when, why and where changes were made with no duplication of common code. Other features include: full DOS pathname, directory, and environment variable support to help manage library maintenance; complete typeset users manual with tutorial; extensive error recovery procedures, support for most programming languages and much more!
SRMS (Version 2.0) $125.00

QMAKE™ is not just another make utility for MS-DOS. Sure it will accept a control file specifying dependancies of your software system, and rebuild your system by compiling only those routines that need to be compiled. It will analyze all dependancies before rebuilding any part of your system, and do it in a way that is extremely fast. Support for macros, multiple entry points, command line parameters, and most languages is also included. QMAKE differs from other DOS make utilites in that it completely integrates with the Software Revision Management System to give you a powerful combination of productivity tools.

QMAKE (Version 1.0).....$ 99.00
SRMS + QMAKE Combination.....$199.95

QUILT COMPUTING

7048 Stratford Road
Woodbury, MN 55125
(612) 739-4650

Circle Number 46 on the Reader Service Card

C CODE FOR THE PC

source code, of course

Concurrent C	$45
Coder's Prolog in C	$45
LEX	$25
YACC & PREP	$25
Small-C compiler for 8088	$20
tiny-c interpreter & shell	$20
Xlisp 1.5a & tiny-Prolog	$20
C Tools	$15

The Austin Code Works
11100 Leafwood Lane
Austin, Texas 78750-3409
(512) 258-0785

Free shipping on prepaid orders **No credit cards**

Circle Number 30 on the Reader Service Card

DeSmet C

now with 32-Bit Pointer Option

C88.....*still* $109

The editors' choice for fast compilation and execution. The **price/performance winner** in all major C benchmarks since 1983. Includes Compiler, Assembler, Binder, Librarian, Execution Profiler and Full Screen Editor. Supports both disk and memory resident Overlays. Contains both 8087 and Software floating point support. Full STDIO library.

Large Case Option.....$50

Makes a great C Compiler even better. Adds 32-Bit Pointers to C88 so you can utilize all of your PC. Groups scalar and static data for fast access. Supports the D88 debugger.

D88.....$50

Gain most of the benefits of an interpreter while losing none of the run-time speed of the C88 compiler. Display C source and variable contents during execution. Set breakpoints by function name or line number. Examine and set variables by name using C expressions.

order direct from:

C Ware Corporation
505 W. Olive, Suite 767, Sunnyvale, CA 94086 U.S.A.
(408) 720-9696 — Telex: 358185
We accept VISA, MasterCard & American Express

Circle Number 38 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

More Optimizing for Speed

by Michael Abrash

In the issue before last, Michael Hoyt optimized an assembler screen dump routine for speed. Michael Eager, James Kane, Lawrence Widing, Bob Smith, Wilbur Street, Bela Lubkin, and Michael Abrash all sent in further optimization of the routine.

Steve Baker added that since the routine is a dump to a printer, the whole process is likely to be bound by the printer's speed, anyway. We all probably thought of that, but it's good to work on the problem all the same.

Here's Michael Abrash's contribution.

In the March/April 1986, issue of *Programmer's Journal*, Michael Hoyt presented a valuable example of high-performance assembly language code. His article should be very useful for beginning and intermediate programmers seeking to improve their code. I would like to present some additional observations, based on several years of IBM PC experience, about optimizing these routines. Although Hoyt's routines are indeed efficient, we will see that it is still possible to improve their performance by more than four times without sacrificing any functionality at all.

A Quick Refresher

Hoyt's article concerned optimization of a routine to double each bit of a byte value in AL and place the result in AX. All other registers were preserved. Hoyt specified that speed was the primary consideration, but showed one way in which size could be reduced if necessary. We will explore both types of optimization to this routine, but speed is our primary consideration. For more information about the routine, refer to "Optimizing for Speed," *Programmer's Journal*, Vol. 4, No. 3, March/April 1986.

Real versus Theoretical Performance

In optimizing the bit-doubling routine, Hoyt relied on published execution times. For the sort of register-intensive code used in the bit-doubling routine, these times are correct but not necessarily accurate. The difference lies in the architecture of the 8088.

The 8088 actually consists of two separate sections. The first section is the Bus Interface Unit (BIU), which, among other things, prefetches up to four future instruction bytes. This means that when one instruction is finished, the next instruction may be already prefetched into the 8088, ready to be executed without the delay associated with executing a memory read. The second section of the 8088 is the Execution Unit (EU), which executes the instructions fetched by the BIU.

The key here is that the instruction timings provided by Intel are for the EU only; that is, if an instruction is already prefetched into the 8088, it will take the specified time to execute; however, if the next instruction is not yet in the 8088, it must be read from memory before it can be executed and the memory read time (4 cycles per byte) is added to the specified execution time. This arrangement generally works pretty well, partly because the 8088 is rather inefficient at calculating the effective address specified by instructions that access memory and instruction bytes can be prefetched during this calculation time. As a result, the 8088 doesn't spend too much time waiting for instruction bytes to be fetched when executing normal code.

Assembly language optimization does not produce normal code, however, in that the most efficient assembly-language code according to Intel's specs uses only registers, avoiding memory accesses as much as possible. Hoyt's final driver, for example, has over thirty consecutive register-only instructions. Alas, instruction fetching means that code such as this does not run at anything like its advertised speed. The highly efficient register-only instructions are executing in 2–4 cycles, but the BIU is taking 4 cycles to fetch each instruction byte. Consequently, after a few shifts or rotates, the queue is emptied and the EU must wait until the next instruction is fetched to resume execution. For SHR and RCR, this means that 8 rather than 2 cycles is the real execution time since both are two-byte instructions. As a result, Hoyt's optimized code is not nearly as fast as it should be. (On an 8086 or 80286, which fetch instructions a word at a time, Hoyt's code would run much faster.)

THIS PREFETCH EFFECT IS VERY REAL. I benchmarked Hoyt's routine Versions 1 through 3, and instruction prefetch took its toll. Version 3 does not show a 37.6% improvement over Version 1 by virtue of using shifts and rotates exclusively—in fact, Version 3 is slower than both Versions 1 and 2 because each instruction after the first few is actually taking 8 cycles rather than 2 to execute.

For example, Version 2 is faster than Version 3 because Version 2 doubles one bit as follows:

```
    SHR BL,1  ;should be 2 cycles; is 8 cycles since 2 bytes
    JNC NEXT_BIT ;should be 16 cycles if jump, 4 if not;
                 ;is probably about 24 if jump, 8 if not,
                 ;since 2 bytes
    ADD AX,1  ;should be 4 cycles; is 12 cycles since 3 bytes
NEXT_BIT:
```

so the average execution time is about

$$30 \text{ cycles} = ((8+24) + (8+8+12)) / 2$$

while Version 3 takes 32 cycles per doubled bit since four two-byte instructions are required per doubled bit. This example illustrates the enormous difficulty of estimating precise real execution times. Without knowing Intel's microcode for the 8088, there is really no way to know whether a full four-cycle penalty is exacted for the second byte of the JNC; it all depends on how many cycles the EU takes before it decides it needs the displacement byte. If the EU wants it immediately, the penalty is indeed 4 cycles; if the EU takes 4 cycles to decide it wants it, there is no penalty at all. I assumed the worst case in the above calculations, but I could be off by several cycles per doubled bit. Timing a particular bit of program code is often the only way to determine how rapidly that code really runs. When performing such experimentation, try to minimize the overhead required to measure the code you're interested in so that you can better judge the magnitude of the difference between different approaches. For instance, I used in-line code for all 256 bit patterns rather than a loop because the loop instruction takes 17 cycles, a significant portion of the time we're trying to measure.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

You should, incidentally, be aware that all branching instructions (JMP, CALL, RET) flush the prefetch queue because any bytes prefetched prior to a branch were read from the code right after the branch, which is probably not where the instruction branched to is stored.

This instruction-fetch effect occurs only in highly register-intensive code—one or two register-only instructions in between memory-access instructions should run at full speed. Again, the best way to judge the true performance of speed-critical code is to time it.

Another Approach

Since we've found that the register-only approach does not optimize as well as expected, we should look for another way to speed the bit-doubling routine. One good alternative is the table-lookup approach.

Because there are only 256 possible bit patterns that can be input to the bit-doubling routine, a table with 256 word entries will suffice for an optimized bit-doubling routine. This makes the routine larger, but it is still only 526 bytes long, not a very significant addition to the average program. If necessary, the routine's size can be reduced greatly with a modified table-lookup approach we'll look at later.

Listing 1 shows the table-lookup version of the bit-doubling routine. Table 1 indicates that the table lookup is indeed a much faster approach, at least two-and-one-half times as fast as Hoyt's versions. (Remember that there is overhead code in these test programs, so the differences between the bit-doubling routines themselves are greater than the times in Table 1 indicate.)

A still faster way to use the table-lookup approach is to implement it as a macro and invoke it as in-line code. The 8088 is relatively slow at branching, so considerable speed improvement can result from conversion from subroutine calls to in-line code. Because the table-lookup code itself is compact (only 13 bytes), this is a very practical approach; the table itself need be present only once in the program, and since a CS prefix is used on the table lookup, it is not even necessary to make sure that DS is set correctly. (I'm assuming the use of NEAR procedures, however.) Listing 2 shows a program that measures the speed of the in-line code approach; register BX is no longer preserved in this listing as I prefer to save the registers only when necessary when I'm using macros to maximize speed and minimize size. Listing 3 shows Hoyt's Version 3 routine rewritten as in-line code; this is for comparison only since this routine is too large to be of practical use as a macro.

Table 1 shows that the use of in-line code boosts the table-lookup approach far past the register-only routines. This in-line routine is well over four times as fast as its in-line register-only counterpart and five or six times as fast as the original bit-doubling routine. This driver can be made a little faster still by assuming that DS equals CS and removing the CS prefix from the table lookup; this shaves about another 5% from the execution time.

In short, if speed is the only consideration, always think of a table lookup first.

Balancing Size and Speed

The table-lookup approach can be applied even when space is at a premium. Listing 4 shows a modified table-lookup routine that reduces the lookup table size from 512 to 16 bytes by performing two lookups, one for each half of the byte that is to be bit-doubled. Naturally, this routine is slower than a straight table-lookup routine, but it is still much faster than the register-only approaches and is very compact as a 46-byte (including the 16-byte lookup table) subroutine. It would not make much sense to implement this routine as in-line code because the extra size of the routine as a macro compared to the straight table lookup would partially offset the smaller size of the lookup table.

Conclusions

As we have seen, a deeper understanding of the 8088 processor and some experimentation can result in surprising improvements in assembly language code performance. Michael Hoyt's routines were well-crafted and a result of the extra research and thought that is essential to top-notch assembly language programming. As often happens, even code that seems optimized yields additional savings with a little extra consideration. As Michael Hoyt suggested, if you keep working over your time-critical code, you may find that you can accomplish results you originally would have considered impossible. When it comes to assembly language, the more you understand, the more your PC can do. □

Michael Abrash is Senior Software Engineer for Tseng Laboratories. He is author of Cosmic Crusader, Big Top, and other video games for the IBM PC published by Funtastic, Inc., and author and co-author of many articles and books about the IBM PC family. Readers can write to him at

6 Remy Place
Newtown, PA 18940

Code follows.

Code	Execution Time in Clock Ticks	Ratio to Listing 1
(Hoyt version 1)	367	1.00
(Hoyt version 2)	347	0.95
(Hoyt version 3)	376	1.02
Listing 1 (Table lookup)	141	0.38
Listing 2 (In-line code table lookup with no PUSH/POP)	66	0.18
Listing 3 (In-line code Hoyt version 3 with no PUSH/POP)	296	0.81
Listing 4 (Reduced-size table lookup)	212	0.58

Table 1: Execution Times

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

1

```
;
; ***Listing 1***
; Bit-doubling routine using table lookup, implemented as a subroutine.
;
; Table of all 256 word patterns possibly generated by bit-doubling a byte.
;
double_table    label   word
        dw      00000h,00003h,0000ch,0000fh,00030h,00033h,0003ch,0003fh
        dw      000c0h,000c3h,000cch,000cfh,000f0h,000f3h,000fch,000ffh
        dw      00300h,00303h,0030ch,0030fh,00330h,00333h,0033ch,0033fh
        dw      003c0h,003c3h,003cch,003cfh,003f0h,003f3h,003fch,003ffh
        dw      00c00h,00c03h,00c0ch,00c0fh,00c30h,00c33h,00c3ch,00c3fh
        dw      00cc0h,00cc3h,00ccch,00ccfh,00cf0h,00cf3h,00cfch,00cffh
        dw      00f00h,00f03h,00f0ch,00f0fh,00f30h,00f33h,00f3ch,00f3fh
        dw      00fc0h,00fc3h,00fcch,00fcfh,00ff0h,00ff3h,00ffch,00ffffh
;
        dw      03000h,03003h,0300ch,0300fh,03030h,03033h,0303ch,0303fh
        dw      030c0h,030c3h,030cch,030cfh,030f0h,030f3h,030fch,030ffh
        dw      03300h,03303h,0330ch,0330fh,03330h,03333h,0333ch,0333fh
        dw      033c0h,033c3h,033cch,033cfh,033f0h,033f3h,033fch,033ffh
        dw      03c00h,03c03h,03c0ch,03c0fh,03c30h,03c33h,03c3ch,03c3fh
        dw      03cc0h,03cc3h,03ccch,03ccfh,03cf0h,03cf3h,03cfch,03cffh
        dw      03f00h,03f03h,03f0ch,03f0fh,03f30h,03f33h,03f3ch,03f3fh
        dw      03fc0h,03fc3h,03fcch,03fcfh,03ff0h,03ff3h,03ffch,03ffffh
;
        dw      0c000h,0c003h,0c00ch,0c00fh,0c030h,0c033h,0c03ch,0c03fh
        dw      0c0c0h,0c0c3h,0c0cch,0c0cfh,0c0f0h,0c0f3h,0c0fch,0c0ffh
        dw      0c300h,0c303h,0c30ch,0c30fh,0c330h,0c333h,0c33ch,0c33fh
        dw      0c3c0h,0c3c3h,0c3cch,0c3cfh,0c3f0h,0c3f3h,0c3fch,0c3ffh
        dw      0cc00h,0cc03h,0cc0ch,0cc0fh,0cc30h,0cc33h,0cc3ch,0cc3fh
        dw      0ccc0h,0ccc3h,0cccch,0cccfh,0ccf0h,0ccf3h,0ccfch,0ccffh
        dw      0cf00h,0cf03h,0cf0ch,0cf0fh,0cf30h,0cf33h,0cf3ch,0cf3fh
        dw      0cfc0h,0cfc3h,0cfcch,0cfcfh,0cff0h,0cff3h,0cffch,0cffffh
;
        dw      0f000h,0f003h,0f00ch,0f00fh,0f030h,0f033h,0f03ch,0f03fh
        dw      0f0c0h,0f0c3h,0f0cch,0f0cfh,0f0f0h,0f0f3h,0f0fch,0f0ffh
        dw      0f300h,0f303h,0f30ch,0f30fh,0f330h,0f333h,0f33ch,0f33fh
        dw      0f3c0h,0f3c3h,0f3cch,0f3cfh,0f3f0h,0f3f3h,0f3fch,0f3ffh
        dw      0fc00h,0fc03h,0fc0ch,0fc0fh,0fc30h,0fc33h,0fc3ch,0fc3fh
        dw      0fcc0h,0fcc3h,0fccch,0fccfh,0fcf0h,0fcf3h,0fcfch,0fcffh
        dw      0ff00h,0ff03h,0ff0ch,0ff0fh,0ff30h,0ff33h,0ff3ch,0ff3fh
        dw      0ffc0h,0ffc3h,0ffcch,0ffcfh,0fff0h,0fff3h,0ffffh,0ffffh
;
;Input: AL=byte to be bit-doubled.
;Output: AX=input byte bit-doubled to a word.
;All registers preserved.
;
bit_doubler:
        push    bx
        xor     ah,ah
        mov     bx,ax
        shl     bx,1    ;double BX to look up in word-wide table
        mov     ax,cs:[bx+double_table] ;get doubled bit pattern
        pop     bx
        ret




;
; ***Listing 2***
; Program to time bit-doubling routine using table
;  lookup, implemented as a macro to gain speed benefits
;  from in-line code.
;
stack   segment para stack 'STACK'
        db      512 dup(0)
stack   ends
;
cseg    segment para public 'CODE'
        assume  cs:cseg,ds:cseg
start   proc    near
        jmp     begin
;
;Input: AL=byte to be bit-doubled.
;Output: AX=input byte bit-doubled to a word.
;Note: register BX not preserved.
;
bit_doubler     macro
        xor     ah,ah
        mov     bx,ax
```

2

```
        shl     bx,1    ;double BX to look up in word-wide table
        mov     ax,cs:[bx+double_table] ;get doubled bit pattern
        endm
;
iterations      equ     1000    ;# of times to bit-double all 256
                                ; possible byte values
;
timmsg  db      'Clock ticks: ' ;timing count displayed at end of run
        db      5 dup(?)
timend  db      0dh,0ah,'$'
;
; Table of all 256 word patterns possibly generated by bit-doubling a byte.
;
double_table    label   word
        dw      00000h,00003h,0000ch,0000fh,00030h,00033h,0003ch,0003fh
        dw      000c0h,000c3h,000cch,000cfh,000f0h,000f3h,000fch,000ffh
        dw      00300h,00303h,0030ch,0030fh,00330h,00333h,0033ch,0033fh
        dw      003c0h,003c3h,003cch,003cfh,003f0h,003f3h,003fch,003ffh
        dw      00c00h,00c03h,00c0ch,00c0fh,00c30h,00c33h,00c3ch,00c3fh
        dw      00cc0h,00cc3h,00ccch,00ccfh,00cf0h,00cf3h,00cfch,00cffh
        dw      00f00h,00f03h,00f0ch,00f0fh,00f30h,00f33h,00f3ch,00f3fh
        dw      00fc0h,00fc3h,00fcch,00fcfh,00ff0h,00ff3h,00ffch,00ffffh
;
        dw      03000h,03003h,0300ch,0300fh,03030h,03033h,0303ch,0303fh
        dw      030c0h,030c3h,030cch,030cfh,030f0h,030f3h,030fch,030ffh
        dw      03300h,03303h,0330ch,0330fh,03330h,03333h,0333ch,0333fh
        dw      033c0h,033c3h,033cch,033cfh,033f0h,033f3h,033fch,033ffh
        dw      03c00h,03c03h,03c0ch,03c0fh,03c30h,03c33h,03c3ch,03c3fh
        dw      03cc0h,03cc3h,03ccch,03ccfh,03cf0h,03cf3h,03cfch,03cffh
        dw      03f00h,03f03h,03f0ch,03f0fh,03f30h,03f33h,03f3ch,03f3fh
        dw      03fc0h,03fc3h,03fcch,03fcfh,03ff0h,03ff3h,03ffch,03ffffh
;
        dw      0c000h,0c003h,0c00ch,0c00fh,0c030h,0c033h,0c03ch,0c03fh
        dw      0c0c0h,0c0c3h,0c0cch,0c0cfh,0c0f0h,0c0f3h,0c0fch,0c0ffh
        dw      0c300h,0c303h,0c30ch,0c30fh,0c330h,0c333h,0c33ch,0c33fh
        dw      0c3c0h,0c3c3h,0c3cch,0c3cfh,0c3f0h,0c3f3h,0c3fch,0c3ffh
        dw      0cc00h,0cc03h,0cc0ch,0cc0fh,0cc30h,0cc33h,0cc3ch,0cc3fh
        dw      0ccc0h,0ccc3h,0cccch,0cccfh,0ccf0h,0ccf3h,0ccfch,0ccffh
        dw      0cf00h,0cf03h,0cf0ch,0cf0fh,0cf30h,0cf33h,0cf3ch,0cf3fh
        dw      0cfc0h,0cfc3h,0cfcch,0cfcfh,0cff0h,0cff3h,0cffch,0cffffh
;
        dw      0f000h,0f003h,0f00ch,0f00fh,0f030h,0f033h,0f03ch,0f03fh
        dw      0f0c0h,0f0c3h,0f0cch,0f0cfh,0f0f0h,0f0f3h,0f0fch,0f0ffh
        dw      0f300h,0f303h,0f30ch,0f30fh,0f330h,0f333h,0f33ch,0f33fh
        dw      0f3c0h,0f3c3h,0f3cch,0f3cfh,0f3f0h,0f3f3h,0f3fch,0f3ffh
        dw      0fc00h,0fc03h,0fc0ch,0fc0fh,0fc30h,0fc33h,0fc3ch,0fc3fh
        dw      0fcc0h,0fcc3h,0fccch,0fccfh,0fcf0h,0fcf3h,0fcfch,0fcffh
        dw      0ff00h,0ff03h,0ff0ch,0ff0fh,0ff30h,0ff33h,0ff3ch,0ff3fh
        dw      0ffc0h,0ffc3h,0ffcch,0ffcfh,0fff0h,0fff3h,0ffffh,0ffffh
;
begin:
        push    cs
        pop     ds              ;DS and CS are the same
;set current time to 0 to make timing easy
        mov     ah,1
        sub     cx,cx
        mov     dx,cx
        int     1ah
;repeat for # of iterations
        mov     cx,iterations
iter_loop:
;
x=      0
        rept    100h            ;bit-double all 256 possible byte values
        mov     al,x
        bit_doubler
x=      x+1
        endm
;
        dec     cx
        jz      iter_done
        jmp     iter_loop
iter_done:
;get current time (count of how long since we set the clock to 0)
        mov     ah,0
        int     1ah
;convert the time count to ASCII & store in the time count message
        mov     ax,dx           ;timer count
        mov     bx,offset timend
        mov     cx,5            ;# of time digits
        mov     si,10           ;we'll divide by 10 to get digits
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

3

```
tisoutlp:
        dec     bx              ;prepare for divide
        sub     dx,dx           ;divide by 10
        div     si              ;convert remainder to ASCII
        add     dl,'0'          ;store digit in message string
        mov     [bx],dl
        loop    tisoutlp
;display the time count message (contains time result)
        mov     ah,9
        mov     dx,offset timmsg
        int     21h
;done   mov     ah,4ch
        int     21h
;
start   endp
cseg    ends    start
        end


; ***Listing 3***
; Michael Hoyt's bit-doubling routine version 3 implemented
;  as a macro.
; Replaces bit_doubler in Listing 2.
;
bit_doubler     macro
        mov     bl,al
        mov     bh,al
;
        rept    8
        shr     bl,1
        rcr     ax,1
        shr     bh,1
        rcr     ax,1
        endm
;
        ends


; ***Listing 4***
; Bit-doubling routine using two half-byte lookups to minimize table size.
;
; Table of all 16 word patterns possibly generated by bit-doubling 4 bits.
;
double_nibble_table     label   byte
        db      000h,003h,00ch,00fh
        db      030h,033h,03ch,03fh
        db      0c0h,0c3h,0cch,0cfh
        db      0f0h,0f3h,0fch,0ffh
;
bit_doubler:
        push    bx
        mov     bl,al
        sub     bh,bh
        shr     bl,1
        shr     bl,1
        shr     bl,1
        shr     bl,1
        mov     ah,cs:[bx+double_nibble_table]
        mov     bl,al
        and     bl,0fh
        mov     al,cs:[bx+double_nibble_table]
        pop     bx
        ret
```

;move upper half of AL
; into lower half of BL
; so we can look up upper
; byte of word result
;get upper byte of result
;isolate lower half of AL
; so we can look up lower
; byte of word result
;get lower byte of result

ED/ASM-86

Now with Full Screen Editor Built in!

The first truly integrated Editor, Assembler, Debugger, and Linker is now available from Oliver Computing Company.

*** The Editor supports block move, block copy, partial save and partial load insertion. All text is tokenized for both efficiency of storage and very fast assembly time. FAST screen I/O and separate editor/debug windows.**

*** The Assembler can be immediately invoked after any editting command. Output can be directed into memory, to a .COM file, .EXE, or to ED/ASM-86's own efficient .LNK file format. 8087/186/286 instructions supported. Macros and most standard pseudo-ops are supported.**
UNIQUE to ED/ASM is its structured assembly language capability. No more wasted labels and time! Multiple segmented programs can be be sent to .COM files.

*** The built in debugger supports symbolic disassembly, integration with the assembler, as well as disassembly of 8087/186/286 instructions. Display 8087 registers in scientific decimal format. SPLIT SCREEN debugging with a dynamic register window is easily invoked. Disassemble programs to disk file for easier analysis. "Single-step" data editting supported for bytes, words, double words, and 8087 floating constants in memory. You can "patch" assemble a line at a time with symbols from previous assembly (even define symbols as you go!)**

*** The linker supports ED/ASM-86 link files, which can be created from virtually any valid ED/ASM-86 source file, even lines with multiple external symbols in expressions. (Note: not compatible with usual link format)**

*** And many more features too extensive to be listed here.**

All of these features exist in one integrated program. A typical development cycle with ED/ASM-86 consists of entering a program with the editor, or editting an existing program, typing "A:,-IM" to assemble into memory, going into the debugger to immediately test the program. Then you go back to the editor for the next cycle.
ED/ASM was developed for Oliver Computing's own use, and has been used for over a year to enhance itself!

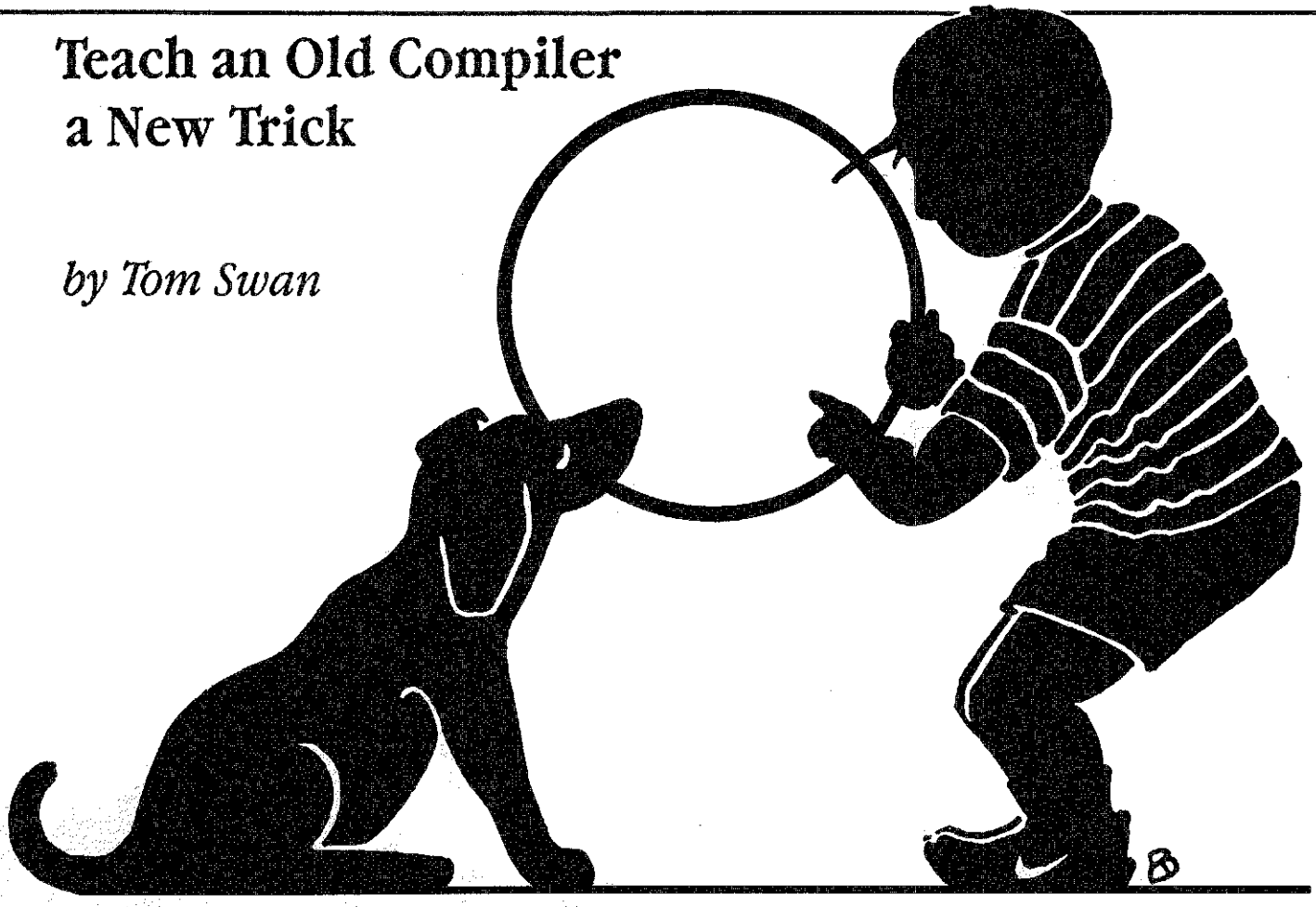
ED/ASM-86 is only $95. Send check or M.O. to:
Oliver Computing Company, P.O. Box 90140, Indianapolis, IN 46290 { (317) 849-4450 }
for immediate delivery via UPS.

Circle **15** on the reader service card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Teach an Old Compiler a New Trick

by Tom Swan



Whenever I read the pronouncement, "You can't do y in language z," it reminds me of that classic comedy record by Bert and somebody or other where Bert, in heavy Bostonian, explains to a stranger how to get to the next town. After a few attempts ("You go down to the corner and . . . no, that's not it"), Bert concludes, "Well, I guess you just can't get theah from heah."

Well, according to the Turbo Pascal manual, you can't pass procedures as parameters to other procedures, a feature of Niklaus Wirth's standard Pascal. But with a simple trick and the help of a "switcheroo" procedure I'll describe in a moment, you can add procedural parameters to Turbo and teach your old compiler a new and useful trick.

Information Hiding

You probably are familiar with the concept of information hiding. A hidden data type's structure is unknown to the program and often to the programmer as well. A good example of information hiding is the Pascal file type. When you declare file f of type FILE, you normally can ignore the internal structure of the file variable. (I'm not talking about the data in the file but rather about the structure of the file variable itself.) Because of information hiding, you can transport your program to other systems where a FILE variable's structure might be radically different. Because such details are hidden from view, the program operates unchanged.

Procedural parameters hide information in a slightly different way. In this case, the compiler hides a process—a procedure or function—instead of a data structure. By doing this, the program can activate a process without having any knowledge of what it does, how it works, or what it is named.

An Example

A simple example explains why procedural parameters are useful. Suppose you are writing a search routine for a database program. The routine should peruse record keys in alphabetic order, calling one of several procedures to process the records defined by those keys. In other words, you want to edit, list, print, and calculate all database records in key order. You might start with the following general form.

```
PROCEDURE Peruse( Option : integer );
BEGIN
   For Key := FirstKey TO LastKey DO
   BEGIN ReadRecord( Key );
      CASE Option OF
         1 : CalcRec;
         2 : EditRec;
         3 : ListRec
      END { case }
   END { for }
END; { Peruse }
```

Although this example is too simple to be of practical use, it demonstrates a problem that often occurs when one procedure

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

must decide which of several others to call. Peruse makes its decision based on Option's value, selecting one of the procedures CalcRec, EditRec, or ListRec in a CASE statement. The main program menu might call Peruse in this way:

```
GetCommand( ch );
CASE ch OF
   'C' : Peruse( 1 );    { CalcRec }
   'E' : Peruse( 2 );    { EditRec }
   'L' : Peruse( 3 )     { ListRec }
END; { case }
```

Although there is nothing wrong with this approach, one thing is clear: Peruse must know in advance all of the processes it might be expected to call. If you want to add a fourth process, you have to add new statements to the main program and to Peruse. To add a PrintRec procedure, for example, you also have to add a new selection in Peruse's CASE statement. There is no way to add new processes without also telling Peruse.

Because of this restriction, Peruse is strongly bound to its host program. This makes it difficult to add Peruse to a general procedure library and merely include or link it to new programs. Instead, you have to insert a new Peruse procedure—and probably rewrite it—for each program that performs database searches.

Adding Procedural Parameters

As you may have suspected me of leading up to, with procedural parameters, you can write a general Peruse that is not tightly bound to its host program. Assuming we have a data type, PROC, which defines a process, and a procedure Call that calls a process of that type, we can rewrite Peruse as follows:

```
PROCEDURE Peruse( Process : PROC );
BEGIN
   For Key := FirstKey TO LastKey DO
   BEGIN ReadRecord( Key );
      Call( Process )
   END { for }
END; { Peruse }
```

The entire CASE statement is replaced by Call(Process). This is the approach we'll follow to add procedural parameters to Turbo. In standard Pascal, by the way, the same procedure would appear in this form:

```
PROCEDURE Peruse( PROCEDURE Process );
```

With its procedural parameter, the new Peruse is no longer tied to its host program. Now, instead of passing a CASE selector, the program simply calls Peruse with the name of the procedure it wants it to perform:

```
Peruse( EditRec );
```

Later, when you add new processes, you do not also have to modify Peruse; therefore, the procedure can go into your library. It will operate unchanged for all programs that need its services. You also can declare new procedures before or after Peruse, which does not need to know the names of procedures it calls. This means you never need to use a FORWARD declaration.

The Compiler Learns Its Trick

How do we go about adding this wonderful ability to Turbo Pascal? After a few failed attempts, I found a way.

First, a few rules are necessary. Procedure parameters must be simple integer addresses representing offsets into the program's code segment. Type PROC, then, is defined as

```
TYPE
   PROC = integer;
```

Procedures that will be passed as parameters must not have their own parameter lists. In other words, all procedures must have the general form,

```
PROCEDURE <name>;
```

Finally, you cannot pass functions; you can pass only simple procedures. Eventually, I'll try to do away with these restrictions, but I wanted to keep the job simple for now.

Having defined the rules, we're ready to write the switcheroo procedure, Call. Its purpose is to activate another procedure whose address is passed as a variable of type PROC. Call is defined as follows:

```
PROCEDURE Call( Process : PROC );
```

To understand how Call operates, you have to look at the compiled code for a typical Pascal procedure and observe what it does to the stack. After examining the stack on entry to a procedure, I discovered a simple way to transfer program flow to a passed address. As defined earlier, when Call begins, the stack contains these two sixteen-bit words:

```
<caller return address>
<process parameter>
```


TURBO PASCAL GENERATOR

THE GTP PROFESSIONAL MODEL

Generate error-free Turbo Pascal source code for:

DATA BASES MENUS
SCREENS REPORTS

Build complete, working programs in minutes!

FEATURES
- Indexed Data Bases
- Multiple Screens
- Automatic Updating
- Built-in Edits
- Retrieval Facility
- Automatic File Build
- Quick Screen Handling
- Speedy DB Access
- Context Sensitive HELP
- Full Keyboard Support

EASY
1. Paint-the-screen
2. Define fields & calculations
3. Generate, compile & RUN

FLEXIBLE
- 100% Turbo Pascal
- Modify in Pascal under Turbo's Editor
- Extensions are easy with 130 page Programmer Reference manual
- 100 documented source code routines included

NOT Copy Protected NO Royalty Fees NO Run-time Library Required

Requires: IBM PC (100% compatible) 256K RAM 2 Disk Drives
PC-DOS 2.0+ Turbo Pascal 3.0 from Borland International

PRICE $200.00
CALL FOR QUANTITY DISCOUNTS
VISA/MC – Check – Money Order – No COD or Purchase Orders
Texas Residents: add $12.25 Sales Tax
Outside US & Canada: add $25.00 Air Postage and make payment by credit card or money order in US Funds.

A E F
ALLEN, EMERSON & FRANKLIN
P.O. Box 928
Katy, TX 77492
(713) 391-8570

Circle Number 53 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

When Call starts running, it executes these instructions, shown here in 8088/86 assembly language:

```
PUSH BP
MOV  BP,SP
PUSH BP
```

After those initial steps, the stack contains two new entries and looks like this:

```
<new bp>                    <-- SP
<caller's bp value>         <-- BP
<caller return address>
<process parameter>
```

With that arrangement, if we simply remove the saved BP register values and swap the caller return address with the process parameter, a subsequent RET will jump to the passed process, which will eventually return to the original caller! In other words, we want the stack to contain

```
<process parameter>
<caller return address>
```

The next RET pops the process parameter, passing control to that address and leaving the caller return address undisturbed. After that, another RET (in the called process) goes back to the caller return address, leaving the stack as it was before Call began. Once Call transfers to another procedure, it never regains control. Remember, Call does not execute a subroutine call—it relinquishes control by way of "returning" to an address on the stack. It's sort of a co-routine switcher. That's why I refer to it as the ol' switcheroo. Call's body requires a simple InLine statement. Here's the finished procedure:

```
PROCEDURE Call( Process : PROC );
BEGIN
   InLine( $89/$ec/   { MOV SP,BP ;reset stack and     }
           $5d/       { POP BP    ; bp registers.      }
           $58/       { POP AX    ;ax<-ret address     }
           $5b/       { POP BX    ;bx<-process address }
           $50/       { PUSH AX   ;reverse the two     }
           $53/       { PUSH BX   ; addresses          }
           $c3  )     { RET       ;execute process     }
END; { Call }
```

The first two instructions reset the stack pointer and BP registers to their values at the start of Call. The POP and PUSH instructions reverse the process parameter with the original return address. After that, RET executes the passed procedure. That procedure's own RET will eventually go back to the statement following the one that activated Call.

How to Use Call

To use the Call procedure, declare a variable of type PROC and assign to it the address of any parameterless procedure. To complete our database example, declare these variables:

```
VAR
   PCalcRec, PEditRec, PListRec : PROC;
```

Assuming you have procedures named CalcRec, EditRec, and ListRec, assign their addresses to the PROC variables with Turbo's OFS function, which returns the offset address in the code or data segments of its parameter. For procedure identifiers, OFS returns the offset from the start of the code segment.

```
PCalcRec := ofs( CalcRec );
PEditRec := ofs( EditRec );
PListRec := ofs( ListRec );
```

After making the assignments, the main program CASE statement calls Peruse by passing the appropriate PROC variable.

```
GetCommand( ch );
CASE ch OF
   'C' : Peruse( PCalcRec );
   'E' : Peruse( PEditRec );
   'L' : Peruse( PListRec )
END; { case }
```

A Final Example

As a full example, the accompanying listing, TestCall, uses Call to activate two procedures, Process1 and Process2. Save the Call procedure in a file named CALL.INC, then type in the example and compile. P1 and P2 are assigned the addresses of the two procedures. The main program calls Process1 with the statement, Call(P1). Then Process1 calls Process2, which calls Process1 again. A global variable, Level, stops the mutual recursion. Usually, one of two such mutually recursive procedures would have to be declared FORWARD, but Call neatly eliminates this requirement.

It would be even better to have full procedural parameters with the ability to pass procedures that have their own parameter lists. Function parameters would also be welcome. Even so, I've found this simple way to add procedural parameters to Turbo Pascal a trick worth teaching an old compiler. □

Tom Swan is the author of several books including Pascal Programs for Data Base Management *and* Mastering Turbo Pascal *published by Hayden Book Company. Address correspondence to Tom at P.O. Box 206, Lititz, PA 17543. Please enclose a self-addressed, stamped envelope if you want a personal reply.*

Code follows.

The Fast Path for New Users of Turbo Pascal...

For TURBO PASCAL On The IBM PC

TRAINING WHEELS ™

- A comprehensive **step-by-step example** to help you get started.
- **30 useful & necessary procedures**—ready to be included in your program—each with source code and with a simple program illustrating its use: Display help screens & menus quickly, check the DOS environment strings, determine where the program was loaded from, manipulate the archive & read-only bits, determine the ROM date & type of machine & type of display being used, retrieve and parse the command line arguments, make your program preserve the color attributes set for DOS, move a file to a different directory without copying it—and many more.
- Helps get your application going quicker and provides examples of how the DOS function calls are used from Turbo.
- **80-page user-guide plus diskette.**

Available for only $25.00
(plus $2 shipping). Cal. Res. add 7%.
Telephone Orders Welcomed (408) 733-4032

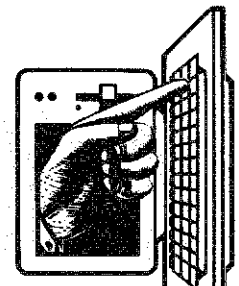
LANDMARK SOFTWARE
1142 Pomegranate Court
Sunnyvale, CA 94087

Circle Number 12 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LISTING

```
PROGRAM TestCall;
{ Test passing procedure parameters }
{ (C) 1986 by Tom Swan ** All rights reserved }

TYPE
   Proc = integer;      { Procedure address }

VAR
   P1, P2 : PROC;
   Level : integer;

{$I CALL.INC}

PROCEDURE Process1;
VAR k : integer;
BEGIN
   Level := Level + 1; k := Level;
   Writeln( ' ':k, 'ENTER procedure 1' );
   IF Level < 5 THEN Call( p2 );
   Writeln( ' ':k, 'EXIT procedure 1' )
END; { Process1 }

PROCEDURE Process2;
VAR k : integer;
BEGIN
   k := Level;
   Writeln( ' ':k, 'ENTER procedure 2' );
   IF Level < 5 THEN Call( p1 );
   Writeln( ' ':k, 'EXIT procedure 2' )
END; { Process2 }

BEGIN
   Level := 0;
   writeln( 'Start of test' );
   p1 := ofs( Process1 );
   p2 := ofs( Process2 );
   Call( p1 );
   writeln( 'End of test' )
END.


{(C) 1986 by Tom Swan ** All rights reserved}

PROCEDURE Call( Process : PROC );
BEGIN
   InLine( $89/$ec/          { MOV SP,BP     ;reset stack and      }
           $5d/              { POP BP        ; bp registers.       }
           $58/              { POP AX        ;ax<-ret address      }
           $5b/              { POP BX        ;bx<-process address  }
           $50/              { PUSH AX       ;reverse the two      }
           $53/              { PUSH BX       ; addresses           }
           $c3       )       { RET           ;execute process      }
END; { Call }
```



Ameritex

The people you've been wanting to meet are online with...

Fast! Friendly! Fun!
At a price you can afford!
Local numbers in over 660 cities.
Call 1-800-322-3633 (914-365-0184 in NY State)

Circle Number 65 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Memory Resident Programs in C

by William Hunt

Memory resident programs such as Prokey or Sidekick are popular in the IBM PC environment. Such programs are first executed from the command line or from a batch file. At that time they install "hooks" (software or hardware interrupts) that allow them to regain control later, reserve the RAM memory area in which they are executing, and then exit to DOS. Later, while a foreground program is running, execution of that software interrupt or occurrence of that hardware interrupt returns control to the memory resident program. After servicing that interrupt, this program restores the context of the interrupted foreground program and allows it to resume execution.

Although it is not obvious from books on C or from C compiler manuals, memory resident programs can be written in C. This article presents the basic tools you need to write such an application. These tools rely on compiler specific information about the implementation of C library functions. Since startup or library source code for Microsoft C 3.0 is not available, the tools shown here apply only to Lattice C, versions 2.15 and 3.00. (The tools should work for Versions 2.10 through 2.14 as well. I could not test the functions with those versions of Lattice C.) The tools work with the Lattice S, P, D, and L memory models. (Use the -mls or-mds forms of the D and L options.) This article covers using software interrupt "hooks"; my book the *C Toolbox* discusses handling hardware interrupts from C.

The tsr.c source file reserves memory and exits to DOS. The tsr function makes a DOS function call (0x31) specifying the number of 16-byte paragraphs to be reserved. If the call is successful, DOS returns control to COMMAND.COM (or the program that executed our program). For versions 2.15 and 3.00 of Lattice C, the __tsize global variable stores the current size of an executing program, including the code, stack, static data, and dynamic memory allocation areas.

Before tsr is called, a memory resident program must call the init__swi function to prepare for handling a software interrupt. This function accepts the number of the interrupt to be serviced and the address and size of a stack area to be used when handling a software interrupt. Since the 8088 stack grows from high to lower addresses, init__swi stores the address of the last word in the stack area. The dptoadr function translates a C data pointer into an 8088 segmented address. Then init__swi makes a DOS call to install the vector for the specified interrupt number. Several interrupt vectors can be installed. The first vector is set to the first entry point address in the swi__adr array, and the second vector uses the second interrupt address in swi__adr.

The assembler source file handswi.asm defines the array of entry point addresses, swi__adr. The assembler language code in this file sets up a proper C environment and calls the C function rcvswint to service the interrupt. When a software interrupt is executed, 8088 register values provide input parameters. The interrupt handler may also return values in the 8088 registers. The assembler code passes a copy of input register values to the C function rcvswint and records any changes to those values. The complete sequence of actions in handswi.asm is

- record which interrupt occurred (0,1)
- save registers on the caller's stack
- change to our stack (the address recorded by init__swi)
- make a copy of input register values on our stack
- create the environment a C function expects (DS,ES,SS,SP,BP registers)
- call rcvswint (int__no,& registers)
- copy register values back to the caller's stack
- change back to the caller's stack
- restore register values (including any alterations)
- return to the caller

The handswi.asm file code works for the lattice S, P, D, and L memory models using the appropriate version of DOS.MAC. The symbols LPROG and LDATA are used to adapt the instructions for 16- or 32-bit code and data pointers. The header file intregs.h defines the register copy structure. Note that the registers included and their order is not the same as the REGS union defined in the dos.h file provided by Lattice.

The prtscn.c file uses these memory resident tools to save screen images. When you press the Shift-PrtSc key combination, the PC's BIOS software generates a software interrupt (INT 5). The normal handler for this interrupt copies chars from the screen to the printer device. Prtscn installs a new handler that copies the characters to a save area. When a different software interrupt (Copy Screen = 0x85) is executed, Prtscn copies this screen image to the address specified by the DS:BX registers. The getscn.c file performs such a function.

The main function in prtscn.c allocates space for saving the screen image. (Although the normal 25x80 screen size requires only 2000 chars, the program allocates 4000 chars in case a larger screen size is defined.)

Then main calls init__swi twice to install the Print Screen and Copy Screen interrupt vectors. A call to tsr keeps the program resident in memory and returns control to DOS.

The rcvswint function uses the interrupt number it receives as an input argument to select the prt__scn or copy__scn functions. The PrtSc interrupt passes no input register values, nor are any values returned in registers. For the Copy Screen interrupt, the DS:BX registers define the address to which the screen image is copied. The number of characters copied is returned in the AX register.

The copy__scn function uses the Lattice library function poke to move characters from the save area to the address specified by the program that executed the Copy Screen interrupt.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The prt_scn function uses BIOS-based Video Output functions to store the screen image. First, a call to vid_state sets the number of columns and the active display page. Next, the current cursor position on the active page is saved. Inside loops over rows and columns, each character is read and stored in the save area. After the contents of the screen have been copied, prt-scn restores the original cursor position.

The program in the getscn.c file executes the Copy Screen interrupt to retrieve the screen image and writes it to a file. The sample output below shows the sequence of operations to install prtscn and place a screen image in a file.

```
C>prtscn =10000              (install the Print Screen Pgm)
C>wp somefile                (execute some other program)
...
[Press Shift-PrtSc]          (prtscn saves the screen image)
...
C>getscn =10000 scn.dat      (places the image into scn.dat)
C>type scn.dat
[ displays the screen image ]
```

When you write memory resident programs, you must follow some extra rules:

1. Close disk files before you call tsr. If you need to do file I/O from the interrupt handler, open the file, perform the read and write operations, and then close the file before returning from rcvswint.
2. Limit the number of files you use. The foreground program may expect to open a number of files.
3. DOS expects one program (process) to be active at a time. You must ensure that a DOS call is not in progress before calling DOS from a memory resident program.
4. Don't use stdin and stdout or DOS calls for console I/O. Use BIOS calls or write directly to display adapter memory instead.
5. Don't call exit or _exit after you have called tsr and later regained control. Some library functions may exit if a serious error is detected. You will have to defeat the C library's error handling. Compile C functions with the -v option to disable stack checking. If a memory program makes a DOS exit call, DOS thinks the foreground program has exited.
6. Be careful in debugging memory resident programs. Normal debuggers don't work well with interrupt handlers. Test individual functions before you test the whole memory resident program.

PC-DOS doesn't know that your memory resident program exists. It thinks that all DOS calls are issued by the foreground program.

There is more to say about writing memory resident programs: defining more than one interrupt handler, handling reentrant calls to our interrupt handler, making DOS calls from a memory resident program, and reclaiming memory used during initialization. □

Bill Hunt writes the Lattice C Users Group *newsletter, where this materal first appeared, and is also the author of the* C Toolbox, *published by Addison-Wesley and currently on B. Dalton's Best Seller list.*

Code follows.

ST-FORTH $49

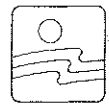
For your IBM PC/XT/AT/PCjr

A complete FORTH development system for beginners or experienced users

- 100% FORTH-83 Standard
- ALL source code provided
  Written entirely in FORTH & FORTH assembler
- Powerful, fast, easy-to-use editor
  Search, locate source, edit errors
- Extensive MS-DOS/PC-DOS interface
  Sequential & random files
- Many more useful utilities
- Supplied as MS-DOS .COM file
  Compatible with all versions of DOS
- 148-page User's Manual
- Unconditional money-back guarantee

Beginner's pkg based on Leo Brodie's *Starting FORTH* available

Call or write for brochure and order form.



Sunset Technology

1954 Menalto Ave.
Menlo Park, CA 94025
(415) 325-3680

Circle Number 60 on the Reader Service Card

Maximum Performance Debugging

DSD86, The PC-DOS Debugger 69.95
DSD87, The PC-DOS Debugger with 8087 Support . 99.95
DSD80, The CP/M Debugger 125.00

SoftAdvances

P.O. Box 49473 • Austin, Texas 78765 • (512) 478-4763

1-800-232-8088

Circle Number 33 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

1

```
/* tsr.c - make terminate but stay resident PC-DOS call */
/* reserves space for code, static data, stack and heap memory pool */
/* the memory pool for dynamic memory allocation should be large */
/* enough for any allocations done after the TSR call is executed. */
/* if the call is successful, tsr() does not return to the caller */
#include "dos.h"
#define TSR_CALL 0x31          /* DOS function code for TSR */
#define CARRY_BIT  0x01        /* carry bit in flags word */
extern unsigned _TSIZE ;       /* total program size */
                               /* maintained by library functions */

int tsr()                      /* issue a terminate but stay */
 {                             /* resident PC-DOS call */
   union REGS sreg , dreg ;

   sreg.h.ah = TSR_CALL ;
   sreg.x.dx = _TSIZE ;
   intdos(& sreg, & dreg ) ;
   else return( dreg.x.ax );      /* return DOS error code */
 }

/* init_swi.c - initialize software interrupt handler */
#include "stdio.h"
#include "dos.h"
#include "intregs.h"

#define SET_VEC 0x25
extern unsigned our_ss ;       /* store address of interrupt stack here */
extern unsigned our_sp ;

extern struct                  /* array of entry point addresses */
 { unsigned intoff ;           /* offset part */
   unsigned intseg ;           /* segment part */
 } swi_adr[] ;                 /* defined in handswi.asm */

int next_vec = 0 ;             /* next vector to assign */

init_swi(int_no,int_stack,sa_size) /* prepare interrupt handler */
 int int_no ;                  /* interrupt number */
 int int_stack[] ;             /* start of stack area */
 unsigned sa_size ;            /* size in words of stack area */
 {
   union REGS sreg , dreg ;
   struct SREGS segs ;
   unsigned ret ;
   char *p ;
                               /* set up interrupt stack address */
   p = & int_stack[ sa_size-1 ] ;/* pointer to last word - high end */
   dptoadr(p,& our_sp, & our_ss) ;/* convert to 8088 seg. address */
   sreg.h.ah = SET_VEC ;       /* install interrupt vector */
                               /* ds:dx = address of next entry point */
   segs.ds = swi_adr[next_vec].intseg ;
   sreg.x.dx = swi_adr[next_vec].intoff ;
   sreg.h.al = int_no ;        /* interrupt number */
   ret = intdosx(& sreg,& dreg,& segs) ;
   next_vec ++ ;               /* advance next available vector */
   return( ret ) ;
 }



 ;  handswi.asm - assembler prologue/epilogue for
 ;  software interrupt handler written in C

 ;  register in/out area
 REGS_SIZE  EQU    28         ; size of area to copy (in words)
 ;  layout of register area
 ;  AX,BX,CX,DX
 ;  SI,DI,BP
 ;  DS , ES
 ;  SP , SS
 ;  IP , CS
 ;  8088 flags
 ;  see intregs.h for C equivalent
```

2

```
    include   dos.mac
;   data section
    DSEG
dstart  equ  dgroup
    public    our_ss,our_sp
our_ss  dw    OFFFFH           ; ss:sp values for interrupt stack
our_sp  dw    OFFFFH           ; initialize to Rom locations

                               ; array of address of entry points
swi_adr dd    intsw0           ; store in 8088 segmented form
    dd    intswl               ; in the data segment
    public    swi_adr
    ENDDS

    if    lprog      eq 1
    extrn     rcvswint:far
    endif

;   code section
    PSEG
    if    lprog eq 0
    extrn     rcvswint:near    ; the C function we call - small code
    endif


;   prologue/epilogue for an interrupt handling function in C

;   record which interrupt occurred here
int_no  dw    0

;   entry points for individual software interrupts
;   init_swi sets interrupt vectors to intsw0 , intswl, ...
intsw0  label      far
    mov   cs:word ptr int_no,0
    jmp   int_common
intswl  label      far         ; entry for second interrupt defined
    mov   cs:word ptr int_no,1
    jmp   int_common
;   can add more entry addresses here for handling more than
;   two software interrupts

    begin     handswi
int_common:
    push ss
    push sp
    push es
    push ds
    push bp
    push di
    push si
    push dx                    ; save the current register values
    push cx
    push bx
    push ax

;   change to our data segment and our interrupt stack
    mov   bx,sp                ; save current stack
    mov   dx,ss                ; in CX & DX for now
    mov   ax,dstart            ; make DS = our
    mov   ds,ax                ; data segment (or group)
    mov   es,ax

    cli                        ; no interrupts while we change stacks
    mov   ax,word ptr our_ss   ; point to our stack segment
    mov   ss,ax
    mov   sp,word ptr our_sp   ; and our interrupt stack
    sti                        ; now allow ints again

;   copy caller's register values to our stack
    sub   sp,REGS_SIZE         ; make room for register values
    mov   bp,sp
    add   bp,2                 ; address of the copy area

    push ds                    ; save segment registers
    push es
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
mov  es,ax              ; point to new stack
mov  di,bp              ; at current location
mov  ds,dx              ; set up addressing for old stack
mov  si,bx
mov  cx,REGS_SIZE       ; now copy registers structure
cld                     ; direction = forward
rep movsb               ; restore DS,ES again
pop  es
pop  ds

push bp                 ; save address of register values
push dx                 ; save caller SS,SP on our stack
push bx

mov  ax,bp              ; save address of register copy
mov  bp,sp              ; mark start of arguments
; direction flag still forward
; rcvswint(int_no,& int_regs) ;
                        ; pass address of reg. values area
if   ldata    eq 1      ; for 32 bit pointers
push ss
endif

push ax                 ; offset part of pointer
push cs:word ptr int_no ; identify which interrupt occurred

call rcvswint           ; call C function

                        ; remove address of registers from stack

if   ldata eq 1         ; 32 bit data pointers
add  sp,6
else
add  sp,4               ; 16 data pointers
endif

;  copy register values to save area on caller's stack stack
   pop  bx              ;  caller's sp
   pop  dx              ;    end    ss

   mov  es,dx           ; set up ES:DI = caller's stack
   mov  di,bx           ;

   mov  ax,ss           ; point DS:SI to register values
   mov  ds,ax           ; on our stack
   pop  si,REGS_SIZE    ; and copy registers values
   mov  cx,REGS_SIZE    ; direction = forward
   cld
   rep movsb

;  restore the caller's stack
   cli                  ; turn off ints. during change
   mov  ss,dx
   mov  sp,bx
   sti

   pop  ax              ; restore old register values
   pop  bx              ; (with any modifications we made)
   pop  cx
   pop  dx
   pop  si
   pop  di
   pop  bp
   pop  ds
   pop  es
   add  sp,2            ; discard pushed SS and SP
   add  sp,2
   iret                 ; end exit
handavi endp
   ENDPS
   end

/* intregs.h - data structure for software interrupt register values */

typedef struct
  { unsigned ax ;  unsigned bx ;
    unsigned cx ;  unsigned dx ;
    unsigned si ;  unsigned di ;
    unsigned bp ;  unsigned ds ;
    unsigned es ;  unsigned sp ;
    unsigned ss ;  unsigned ip ;
    unsigned ca ;  unsigned flags ;
  } INT_REGS ;
```


new! Version 3.0 WIZARD C

The following SIEVE benchmark was run without register variable declarations on an IBM/PC with 640K memory and an 8087.

	Exec Time	Code Size	EXE Size
Wizard C 3.0	**: 6.8**	**130**	**7,766**
Microsoft	:11.5	186	7,018
Lattice	:11.8	164	20,068

The new **Wizard C version 3.0** sets new records for all out speed! It leaves other C compilers in the dust! When your project depends on that last ounce of speed, choose Wizard.

Fast executable code, with multiple levels of optimization.

Six memory Models, supporting up to 1 megabytye of code and data, plus mixed model programming.

Effective error diagnosis, including multiple source file cross-checking of function calls.

A comprehensive runtime library, including fully portable C functions, MSDOS and 8086 functions.

8086, 8087, 80186 and 80286 hardware support.

Full Library Source Code included with package.

ROM based application support, including a ROM development package available to create Intel Hex files.

Fully ANSI compatible language features.

"...The compiler's performance makes it very useful in serious software development."

PC Tech Journal
January, 1986

" Wizard's got the highest marks for support."

"The Wizard Compiler had excellent diagnostics; it would be easier writing portable code with it than with any other compiler we tested."

Dr. Dobb's Journal
August, 1985

"...written by someone who has been in the business a while. This especially shows in the documentation."

Computer Language
February, 1985

(617) 641-2379

Only $450.

WIZARD
SYSTEMS SOFTWARE, INC.
11 Willow Court, Arlington, MA 02174

MasterCard VISA

Circle Number 49 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

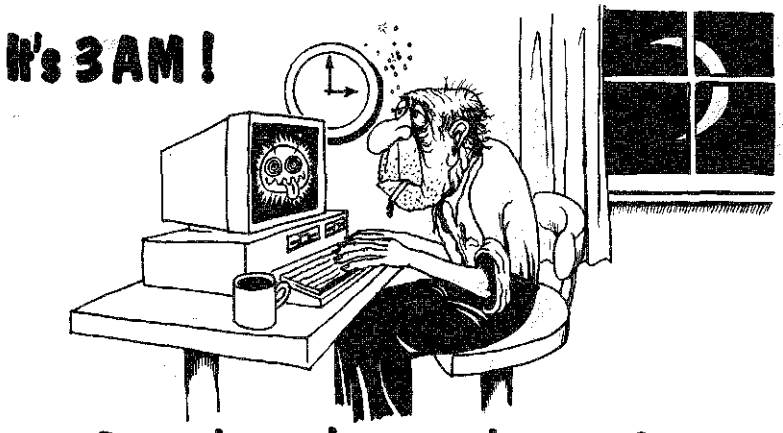
4

```
/* prtscn.c - saves screen image when PrtSc key pressed */
#include "stdio.h"
#include "intregs.h"
#define PrtSc_INT 5          /* Print Screen interrupt number */
#define CopySc_INT 0x85      /* interrupt number for our Copy Screen */
#define SA_SIZE 4000         /* save area size (in bytes) */
#define IS_SIZE 3000         /* stack size (in 16 bit words) */
#define RMAX 25              /* number of rows on the screen */
static int cols = 0 ;        /* number of columns on screen */
                             /* = 0 if no image currently saved */
                             /* set when an image is saved */
char *save_area ;            /* holds address of save area */
char *malloc() ;

main()                       /* installs print screen handler */
 {
   int istack[IS_SIZE] ;     /* use this area for an interrupt stack */

   save_area = (char *) malloc(SA_SIZE) ; /* allocate save area */
   if( save_area == NULL )
        { cputs("can't allocate save area \n") ;
          exit(10) ;
        }

   init_swi(PrtSc_INT,istack,IS_SIZE) ; /* install Prt Scn handler */
   init_swi(CopySc_INT,istack,IS_SIZE) ;/* and Copy Screen Handler */
   ter() ;                   /* terminate-stay resident exit */
 }

int rcvswint(int_no,pireg)   /* called when s/w int occurs */
 int int_no ;                /* which interrupt we are processing */
 INT_REGS *pireg ;           /* address of input register values */
 {
   if( int_no == 0 )         /* check which interrupt */
        save_scn(save_area);/*    PrtSc - save image */
   else                      /*    Copy Screen - copy to DS:BX */
        pireg->ax = copy_scn(save_area,pireg->bx,pireg->ds) ;
 }

int copy_scn(p,off,seg)      /* copy screen image to address given */
 char *p ;                   /* screen image starts here */
 unsigned off ;              /* offset part of segment address */
 unsigned seg ;              /* segment part of address */
 {
   int n ;

   n = RMAX*cols ;           /* number of chars saved */
   poke(seg,off,p,n) ;       /* copy chars to specified address */
   cols = 0 ;                /* and reset to indicate no image saved */
   return( n ) ;             /* return number of chars */
 }

int save_scn(p)              /* copy chars on screen to a save area */
 char *p ;                   /* address of save area */
 {
   char at ;
   int rs , cs ;             /* save original row and column here */
   int r , c ;               /* row and column loop indices */
   vid_state(&cols) ;        /* get number of columns, active page */
   vid_gpos(& rs,& cs) ;     /* save current position */
   for(r=0;r< RMAX ; r++)    /* loop over rows and columns */
        { for(c=0; c<cols ; c++)
             { vid_spos(r,c) ;/* set cursor position */
               vid_gca(p,& at);/* get the char at this position */
               p++ ;         /* advance save area pointer */
             }
        }
   vid_spos(rs,cs) ;         /* restore cursor position */
 }
```

5

```
/* getscn.c - get a screen image from prtscn.c and write to a file */
#include "stdio.h"
#include "dos.h"
#define CopySc_INT 0x85      /* Copy Screen Interrupt number */
FILE *fopen() ;

main(argc,argv)
 int argc ;
 char *argv[] ;
 {
   FILE *out ;
   char scn[4000] ;
   int n ;
   union REGS sreg , dreg ;
   struct SREGS seg ;
   if( argc < 2 )            /* file name specified ? */
        [ cputs("USAGE - getscn filename");
          exit(5) ;
        }

   out = fopen(argv[1],"ab") ;/* open the file */
   if( out == NULL )
        { puts("can't open file");
          exit(10) ;
        }
   dptoadr(scn,& sreg.x.bx, & seg.ds) ;/* get segmented address of */
                             /* area to copy to */

   int86x(CopySc_INT,& sreg,& dreg,& seg) ; /* do the interrupt */
   n = dreg.x.ax ;           /* get number of chars copied */
   if( n > 0 )               /* anything there ? */
        { fwrite(scn,1,n,out);/*  yes - write chars to the file */
          fputs("*** \r\n",out);/*      and a separator */
        }
   else puts("no image");    /*   no  - error message */
   fclose(out) ;             /* then close file */
 }


/* video5.c - Rom BIOS call for Video_IO  - get video state */
#include "stdio.h"
#include "dos.h"
#define VIDEO_IO 0x10        /* interrupt number for VIDEO_IO */
#define V_STATE 15           /* function code for getting video state */
extern int v_page ;          /* page number for read/write calls */

int vid_state(pcol)          /* get current state info */
 int *pcol ;                 /* store number of columns here*/
 {                           /* return the current mode value */
   union REGS sreg , dreg ;

   sreg.h.ah = V_STATE ;
   int86(VIDEO_IO,&sreg,&dreg) ;
   v_page = dreg.h.bh & 0xff ;
   *pcol = dreg.h.ah & 0xff ;
   return( (int) (dreg.h.al & 0xff) ) ;
 }


/* dptoadr2.c - convert data pointer to 8088 segmented address */
#include "dos.h"

int dptoadr(p,poff,pseg)     /* convert data ptr to 8088 address */
 char *p ;                   /* C pointer to data */
 unsigned *poff ;            /* put offset part of address here */
 unsigned *pseg ;            /* put segment part of address here */
 {
    struct SREGS sr ;

    if(sizeof(char *) == 2) /* Is the pointer  16 bits long ? */
      { segread(&sr) ;       /*   y - use current data segment */
        *pseg = sr.ds ;      /*       value */
        *poff = (long) p ;   /*       and use pointer as offset */
      }
    else                     /*   n - (32 bits)  */
      { *poff = *   ( (unsigned *) &p) ; /* offset part */
        *pseg = * ( ( (unsigned *) &p) + 1) ; /* segment part */
      }
 }
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



Do you know where your bugs are?

This C programmer is finding his bugs the hard way...one at a time.
That's why it's taking so long. But there's an easier way. Use

PC–Lint

PC–Lint* analyzes your C programs (one or many modules) and uncovers glitches, bugs, quirks, and inconsistencies. It will catch subtle errors before they catch you. PC–Lint resembles the Lint that runs on the UNIX* O.S., but with more features and some awareness of the 8086 environment.

- Full K&R C
- Supports Multiple Modules–finds inconsistencies between declarations and use of functions and data across a set of modules comprising a program.
- Compares function arguments with the associated parameters and complains if there is a mismatch or too many or too few arguments.
- User-modifiable library description files for most major compilers.
- All warning and information messages may be turned on and off globally or locally (via command line and comments) so that messages can be tailored to your programming style.
- All command line information can be furnished indirectly via file(s) to automate testing.
- Use it to check existing programs, programs about to be exported or imported, as a preliminary to compilation, or prior to scaling up to a larger memory model.
- All one pass with an integrated preprocessor so it's very fast.
- Has numerous flags to support a wide variety of C's, memory models, and programming styles.
- **Price: $139.00 MC, VISA**
  (Includes shipping and handling) PA residents add 6% sales tax. Outside USA add $10.00
- Runs under MS-DOS* 2.0 and up, with a minimum of 128Kb of memory. It will use all the memory available.

Trademarks: PC–Lint (Gimpel Software), UNIX AT&T), MS-DOS (Microsoft)).

NEW!! **Amiga - Lint**
Special Introductory Price
$98.00

GIMPEL SOFTWARE
3207 Hogarth Lane • Collegeville, PA 19426
(215) 584-4261

Circle Number 11 on the Reader Service Card

C DYNAMO

FREE! PCUNIX™

(WITH ANY PURCHASE OF $250 OR MORE
LIMITED TIME OFFER)

LIBRARY: *325 fully tested functions screen handling/graphic, cursor/ keyboard/data entry, string handling, status and control. utility/DOS/BIOS/ time/date, printer control, 4 disks* *$129.95*

WINDOWS: *The most powerful windowing package available, overlays, borders, pop-up menus/ help windows, zap instantly on/off screen, status lines, horizontal/ vertical scrolling, color control/ highlighting, word-wrap, files to windows, keyboard to windows 3 disks* *$129.95*

SUPERFONTS FOR C: *Dramatic, high impact screens. Use our character & image libraries and functions, or create your own. Font & Function Library* *$49.95*

B-TREE LIBRARY: *Fast indexing B-tree. 16 million each: keys and records, unlimited keys per file, variable length records. PC-DOS, UNIX, XENIX, CP/M.* *$79.95*

ISAM DRIVER: *$49.95*

COMPATIBILITY: *PC/XT/AT; Full K&R, CI-C86/De Smet/ Lattice/ Microsoft 3.0/Aztec/others.*

VALUE: *All source code, No royalties. Best documentation available.*

No matter what else you have, you need these.

C-TERP: *Simply the best C interpreter available. See C Journal Summer 1985*
C-TERP (specify compiler) *$299.95*

PCUNIX: *Multi-tasking, networking multi-user. With source code* *$99.95*
™Wendin Software

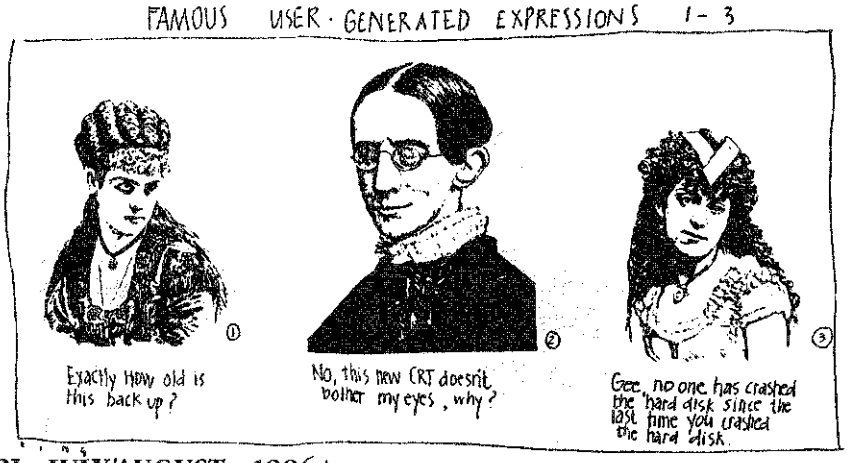
COMBINE AND SAVE!

C LIBRARY plus C WINDOWS
BOTH for only $179.95
+ SUPERFONTS FOR C . $199.95
+ B-TREE and ISAM $299.95
(A $440 VALUE)
C LIBRARY plus C WINDOWS
+ SUPERFONTS
+ C-TERP (a $610 value) $459.95
All (a $740 value) $549.95

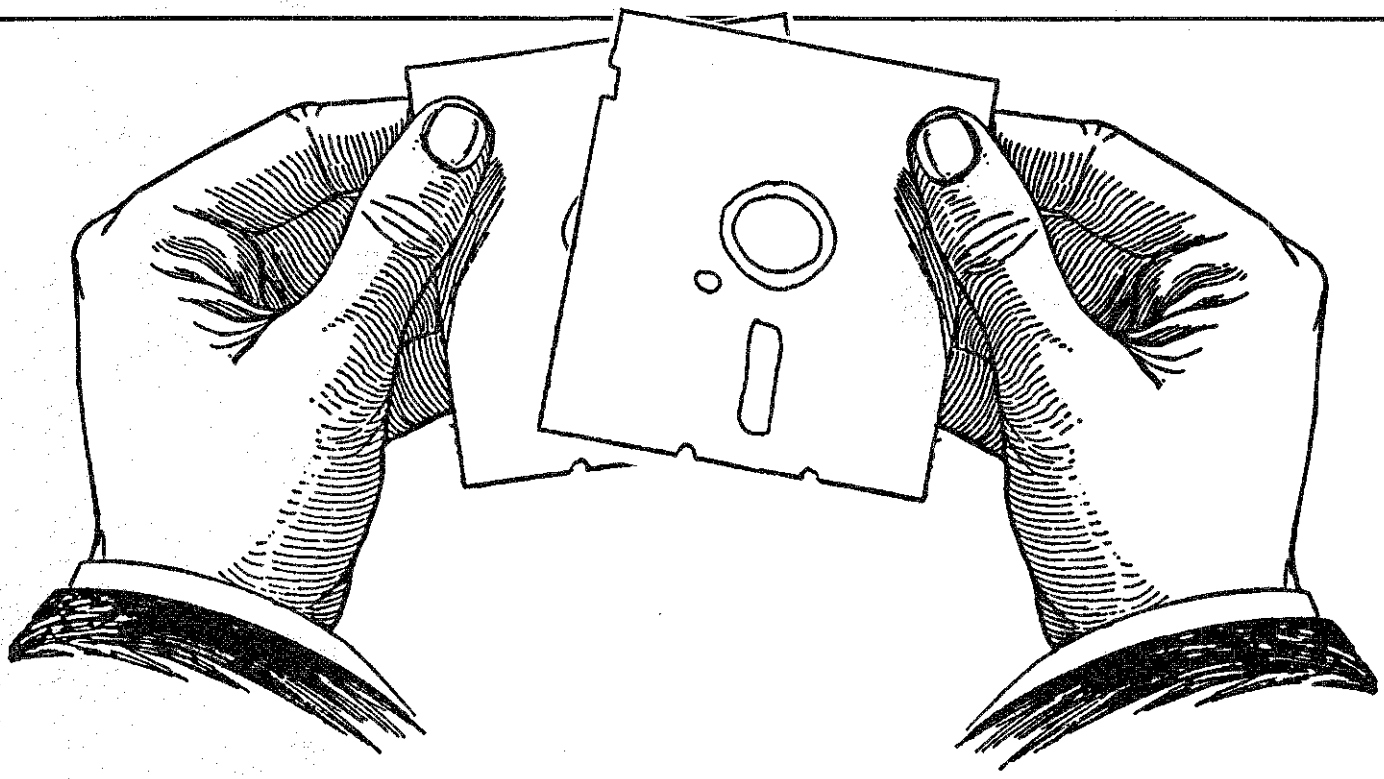
Entelekon

12118 Kimberley, Houston, TX 77024
713-468-4412
VISA MASTERCARD CHECK COD

Circle Number 9 on the Reader Service Card


FAMOUS USER-GENERATED EXPRESSIONS 1-3
Exactly how old is this back up?
No, this new CRT doesn't bother my eyes, why?
Gee, no one has crashed the hard disk since the last time you crashed the hard disk.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



Tips on Formatting with BASIC PRINT USING Statements

by Bernard H. Robinson, Jr.

The PRINT USING statement is the prototype of a family of three statements, the other two being LPRINT USING and PRINT # USING. These statements are useful because of the degree of control they give over the printing of string and numerical expressions. They are especially helpful in formatting numerical data into neat columns with decimal points aligned, rounding numerical data, adding a dollar sign to dollar amounts, pointing off large numbers with commas, etc. The IBM BASIC manual discusses the PRINT USING statement well enough to get a beginner started using it, but it is only through much trial and error that many of the finer points of using this statement are discovered. I hope this discussion will help pave the way for those who are just learning to use this family of BASIC statements and maybe even offer those who have been using them right along a useful tidbit or two. The emphasis is on the most common applications of the statements.

The PRINT USING statement writes to the screen, the LPRINT USING statement writes to the printer, and the PRINT # USING statement writes to a file. Otherwise, the statements are almost identical.

Introduction to PRINT USING Statements

The following is an example of a simple PRINT USING statement:

```
PRINT USING "The yearly total is: $$####.##_."; tval
where "tval" is a numerical variable.
```

The format of the PRINT USING statement is

```
PRINT USING F$; list [;];;
```

F$ represents either a string constant or a string variable that defines the special format to be used in printing the expressions in the list part of the statement.

Notice that a semicolon must follow the formatting string.

list represents a list of one or more expressions, either string or numeric, to be printed as defined by F$. Semicolons or commas are used as separators between expressions.

[;] is an optional semicolon or comma, used when the next PRINT or PRINT USING statement is to continue on the same line. If the final semicolon or comma is omitted, BASIC appends a carriage return and a line feed.

Just to be very clear about it, when I say that the formatting string F$ defines a format, I mean by "format" everything that F$ defines. As we will see, the formatting string can define quite a lot. When I refer to a "field" in the format, I refer to a group of formatting characters defining how an expression is to be printed. Aside from these fields, the formatting string may contain string constants which are also printed, such as the string "The yearly total is: " in the example given above. The formatting string is constructed using ASCII characters, some of which have special meaning in this string. The rules for constructing formatting strings are discussed in some detail below.

IBM PC BASIC allows more than one print field to be defined in a single statement although all the examples in the BASIC manual show only one field per statement. Multiple field definitions in

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

a single PRINT USING statement will be discussed below.

When only one field is defined in a statement, all of the expressions in the list are printed using the defined format, including any string constants in the format string. Quite often a single line of printed output will require a number of PRINT USING statements to define the various fields desired. Since strings and numerical expressions do not use the same formatting characters, the expression list of a statement with only one field definition must be either all string expressions or all numerical expressions, depending on the type of field defined. Although a semicolon is required immediately after the format string, commas are acceptable as separators between expressions and at the end of the statement to indicate continuation on the same line. Unlike the PRINT statement, the PRINT USING statement treats comma separators between expressions exactly as it does semicolons. It may be easier to use only semicolons with PRINT USING statements rather than trying to remember where semicolons are required and where commas are allowed.

Specifying String Fields

Strings as long as 255 characters (excluding the final carriage return and line feed added by PRINT USING) can be output. The following characters are used to define fields for strings: !, \, and &.

An exclamation point (!) means to print only the first character of a string expression. It could be used to cause a string containing a person's first name to be printed as only the initial, for example. The backslash (\) is used to define string character fields of two or more characters; two backslashes with no space between them means to print two characters. Two backslashes with one space between the backslashes means to print three characters. More spaces are used between the backslashes to define longer fields. The string expression characters are always left-justified in the field, and any characters not fitting in the field are ignored. If the string is shorter than the field, the field is finished out with spaces.

The "&" character means to use a variable width field as required to print out the string expression exactly as it is, including any spaces in the string expression.

The Escape Character

The underscore character (_) is an escape character for both string formatting characters and numerical formatting characters. When it is included in the format string just before a formatting character, it means that the next character is to be interpreted as a literal ASCII character instead of as a formatting character. For example, if you wish to print the character "!" after an expression to be printed in the "&" format, the format would be given as "&_!". Without the underscore character before the "!" character, BASIC would interpret this as the definition of two fields in the statement. To print an underscore character as part of a string constant included in the format string, put two underscores in sequence.

Although the period and comma are used as special format characters in numerical format strings, they are allowed at the end of the formatting string without the escape character preceding them in order to print a period or comma immediately after the numerical data. However, it does no harm to put the escape character before them in this case, and it is probably a good idea to do so to make it clear to anyone reading the program that they are to be taken as literal characters. The escape character does not request an additional space in the field.

Special Characters in Numeric Data Formatting

PRINT USING statements are used more often to format numerical expressions than for string expressions. The following characters have special meaning in printing numerical expressions. Their meaning is summarized here and explained in more detail below:

#	Number sign, primary numeric formatting character.
.	Decimal point, specifies the decimal position.
+	Plus sign, controls plus and minus sign printing.
–	Trailing minus sign, prints minus sign at end of numbers.
**	Double asterisk, fills leading spaces with asterisks.
$$	Double dollar sign, prints a dollar sign.
**$	Double asterisk with dollar sign, combines ** with $$.
,	Comma, points off large numbers with commas.
^^^^	Four carets, requests an exponential numeric field.
_	Underscore, escape character, as for string formats.
%	Percent sign, field overflow indicator (not used in format strings but is output on overflow; see below).

Maximum Length of Numeric Fields

No more than 24 digits may be specified in any numeric field.

Integer Field Definition

The number sign (#) is used in essentially all numeric format strings. For printing positive integers, one "#" character is put in the field definition string for each digit place in the largest number to be printed. If negative numbers of the same magnitude are to be printed, an additional "#" character is required to allow space for the negative sign to be printed. For example, the string "###" defines a field for printing integers from –99 to 999. The integer is right-justified in the defined field and padded on the left with spaces, if required. If the defined field is too small to contain the value of an expression to be printed, including the minus sign for a negative expression, a field overflow will occur. See below for other field options for printing plus and minus signs, dollar signs, etc. and overflow handling.

Overflow Handling

In any numerical format, when a number is too large numerically to be printed in the field defined for it, it is still printed in full, but is printed with a "%" sign before the number to indicate the overflow. This is a safety feature: The printed value will be longer than the format specified, which may spoil the appearance of the output, but this will avoid the printing of truncated numeric data that could be mistaken for complete data.

Fixed Point Field Definitions

The decimal point character is combined with the number sign characters to indicate the number of digit positions to be used on both the left and right of the decimal. For example, the field definition string "###.##" means that two positions to the right of the decimal point will always be printed (BASIC rounds the numbers to two places and/or supplies zeros to fill the spaces to the right of the decimal point, as necessary) and up to 3 character positions on the left of the decimal point will be printed. The printed number is padded on the left with spaces to fill out the field, if required. The decimal character in the field definition string requests another character in the field, which can be the space to print the decimal.

This paragraph applies to the handling of fixed point numbers with an absolute value of less than one: If there is more than one

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

"#" to the left of the decimal point in the field, BASIC will always supply a zero to the left of the decimal. If there is only one "#" to the left of the decimal in the field, a zero will be printed to the left of the decimal for positive numbers, but only a minus sign will be printed there for negative numbers.

See below for other options for printing plus and minus signs, dollar signs, etc.

Printing Plus and Minus Signs

The default printing of signs has been described above (plus signs are not printed, but minus signs are printed). To secure other sign printing options, the plus sign and the minus sign are used as control characters in the format fields. Placing a plus sign in the field string at the beginning of the field requests printing the required sign, either plus or minus, before the first printed digit of the numeric expression. The plus sign in the field string allocates another character to the field that can be regarded as the space for the sign. CAUTION: This option cannot be used in combination with the option to print dollar signs. See below.

A plus sign can also be placed at the end of the field string. This asks BASIC to print the required sign, plus or minus, after the number. The plus sign in the format string requests an additional character position in the field, which can be the position for the printed sign. It is okay to use the option to print dollar signs with this option.

A minus sign is allowed in the field definition string only at the end of the string, where it means print a minus sign following the number if it is negative and a space if it is positive. The minus sign in the format allocates another character to the field, which can be the space for the sign. It's all right to use this format with the "$$" option. See below.

Filling Blank Spaces with Asterisks

Starting the field string with two asterisk characters requests that leading spaces in the field be filled with asterisks. Each of the two asterisks in the format string requests an added print position so that if a format were originally "###.##" and you wanted to change to this option, you would change it to "**#.##". This option is sometimes used to make it more obvious that a printed value has been altered after printing.

Printing Dollar Signs

Two dollar signs beginning the field definition are used to request that a dollar sign be printed before the first digit in the field. Each dollar sign in the format requests an added print space in the field, one of which will be needed to print the dollar sign; so "###.##" would become "$$###.##" to allow the same range of numeric print, and the field would be one space wider than before. CAUTIONS: This option is not allowed with numbers in the exponential form. For negative numbers, this format can be used only if the minus sign is requested to be printed at the end of the number (Example: "$$##.##-").

The characters "**$" at the beginning of a field definition requests a dollar sign as in the "$$" format with any extra leading spaces filled with asterisks as in the "**" format. These characters in the field string request 3 spaces, so the number of "#" characters must be determined accordingly. CAUTIONS: See the cautions above for the "$$" format. This format seems to be most useful for applications such as writing in the amount of checks.

Pointing Off with Commas

A comma placed in the field definition just before the decimal point requests the printing of a comma every third position to the left of the decimal point. For example, "######,.##" would print the number 100000 as "100,000.00". The comma in the format string requests an added space, which can be the space to print the comma. Additional "#" characters are needed in the field if the expression printed is large enough to require more than one comma in printing.

Exponential Field Definitions

Four carets (^^^^) are used to force the output to be in the exponential form. The four carets represent the four spaces required to print the "E" or "D" and the exponent (with sign) required by the exponential format. (BASIC uses the "E" when printing single precision expressions and "D" when printing double precision expressions.)

The preferred way to use this option is to specify the field with two "#" characters before the decimal point (for example, "##.######^^^^"). The number of "#" characters to put in the field definition to the right of the decimal is determined by the precision with which the number is to be printed. The reason that the two "#" characters are preferred on the left is that this results in the printing of numbers in the standard exponential format for scientific and engineering work. BASIC adjusts the exponent of the printed value so that one, and only one, non-zero digit is printed to the left of the decimal. A leading space is printed for positive expressions, and a minus sign for negative expressions.

As an alternative to the above, specifying "+#" to the left of the decimal gives the same results except that the sign, plus or minus, is always printed.

Occasionally someone may want to print numerical data with the exponent adjusted such that the number to the left of the decimal is always zero. (Some FORTRAN programs format numbers this way with their "E" format, and someone might wish to simulate this.) This can be accomplished by placing only one "#" character to the left of the decimal point in the field definition instead of the two recommended above. The zero is printed for positive numbers, and only a negative sign appears to the left of the decimal for negative expressions. An alternative is to use only a "+" to the left of the decimal in the field definition: This prints only the sign, positive or negative, before the decimal.

Although the PRINT USING statement allows more "#" characters to appear to the left of the decimal in the exponential format and will adjust the exponent on the printed number to print more characters to the left of the decimal, such formats are not likely to be of practical value.

CAUTION: The dollar sign option cannot be used with this option.

Defining More Than One Field in a PRINT USING Statement

When more than one print field is specified in a PRINT USING statement, BASIC prints the first expression in the list using the first print field, the second using the second print field, etc. If there are more expressions in the list than there are print field definitions, BASIC starts over again at the beginning of the format string. If the format string includes string constants, they too are printed again when BASIC starts over at the beginning of the string. When all the expressions in the list for a statement have been printed, BASIC ignores any unused print fields in the statement. Each expression must be of the same type as its corresponding field (string or numeric). With appropriate field definitions, both string and numeric expressions may be included in the same PRINT USING statement.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Example Program with PRINT USING Statements

The program listing P-USG.BAS is a somewhat contrived demonstration of formatting output two ways. The data is first printed with PRINT USING statements containing only one field definition per statement. The same data is then formatted using PRINT USING statements with multiple fields defined in each statement. Some of the advantages of each approach will be apparent from studying the listing.

The program output is shown in Figure 1. The program prints out six columns of numerical data with column headings and a brief summary message. The amount of data printed was limited so that a 25 line, 80 column screen would show the entire printout at once. Just for the demonstration, the program assumes that a dollar amount corresponding to each month has to be multiplied by a scale factor, which varies from month to month, to compute a net amount for each month. The monthly net amounts are accumulated in column six. Finally, at the end of the printout, a summary consisting of the total of the raw amounts and the average monthly scale factor is printed out. A few points of interest about this example program will be discussed.

Using TAB( ) Functions to Position Columns of Data

In the case of using only one field definition per statement, extensive use is made of TAB( ) functions. The following tip is just another instance of the programming principle that it is often advantageous to use a variable in place of a constant that may appear in more than one place in a program. In line 1010 of the program the variables c1 through c6 are assigned values. These represent the column locations for columns 1 through 6. They are used in the TAB( ) functions in print statements (lines 1050,1070, etc.) to tab to the print positions for the three lines of column headings and in the printing of the numerical data. Actually, c1 is not used since the first column starts in column one, but it is included to define all column positions in one handy line of the program. Sometimes you may want to shift the position of a whole column to the left or right. Assigning a new value to the position variable for the column will shift the position of the whole column with its heading. Larger values shift to the right, and smaller values to the left. In another case, you might want to shift the heading left or right without moving the numeric data of the column. For example, TAB(c4 + 2) for locating the print position for a column 4 heading would shift the heading of the column on that line by only two spaces to the right without moving any other part of the printout. This use of TAB( ) functions and column position variables can be a big help in coordinating the various print positions.

Notice that the PRINT statement, used for the TAB( ) functions, can be mixed freely with PRINT USING statements in printing one line of output.

The "&" Format vs. the "\ \" Format

In formatting the column headings, I sometimes used the "&" format (see lines 1040 and 1060, for example) and sometimes the "\ \" format (see lines 1140 and 1160, for example). Either format can be used to print string constants, but the "&" format is easier to use because you do not have to change the number of spaces between the two back slashes if you change the string constant to one with a different number of letters.

Maxine shudders to realize the senior programmer does not understand the concept of discrete data.

TIME...

If you're a programmer, computer time is your stock-in-trade. The time you spend on software development is vitally important to your livelihood. Spend too much and you work for peanuts, spend too little and you price yourself out of a job.

If there were a way to monitor how much time you spent on computer tasks, a way to find out how closely your estimated project time compares to reality, a way to bill out all of your time, wouldn't you?

With SYSLOG you can.

SYSLOG requires an IBM® PC/jr/XT/AT, COMPAQ™, or other MS-DOS compatible computer with 128K memory, one disk drive, Display, and a printer. Also requires DOS 2.0 or greater.

Only $59.95

Shipping/Handling
Within USA add $3.00
NY Residents add sales tax

Ask your dealer or order direct. Credit cards accepted; most orders shipped within one business day.

SYSLOG
SYSTEM LOG UTILITY

APEX RESOURCE
23 Christine Court
Stormville, NY 12582
914-221-2611

Circle Number 54 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Plain PRINT statements could have been used to print the headings of this program. If string variables had been involved in the headings, PRINT USING statements would have been more useful.

Using Multiple Field Definitions

The subroutines at 3000, 3100, 3200, and 3300 print the data using multiple field definitions in each PRINT USING statement. The heading is printed using only PRINT statements. Because the program was to be printed in a magazine, each line was limited to less than 80 characters. If longer lines had been used, still fewer statements would have been required to print the data. Using long names for variables limits the usefulness of multiple field definitions (fewer variable names can be typed on a line). With multiple field definitions, it may be a little more difficult to align the columns and headings as you prefer them.

Double Precision Variables

The DEFDBL statement on line 100 declares all numerical variables starting with "D" (all numerical data variables in this program) to be double precision variables. If this line is deleted and the program is rerun, the effect of not using double precision variables will be seen. In this program, little difference will be noted. In a program using larger numbers, the difference may be considerable. It may be instructive to try this with your own programs. □

Figure 1. Output from P-USG.BAS

LINE NO.	MONTH	RAW AMOUNT	SCALE FACTOR	NET AMOUNT	ACCUM. AMOUNT
1	JAN	$9,645.35	0.99845	$9,630.40	$9,630.40
2	FEB	$8,861.45	0.89637	$7,943.14	$17,573.54
3	MAR	$7,579.23	1.04578	$7,926.21	$25,499.74
4	APL	$8,472.39	0.99563	$8,435.37	$33,935.11
5	MAY	$9,384.74	0.98724	$9,264.99	$43,200.10
6	JUN	$9,381.36	1.00376	$9,416.63	$52,616.74

SUMMARY, TOTAL RAW AMOUNT AND AVERAGE SCALE FACTOR TO DATE: Raw Amount: $53,324.52. Scale Factor: 0.98787.

LINE NO.	MONTH	RAW AMOUNT	SCALE FACTOR	NET AMOUNT	ACCUM. AMOUNT
1	JAN	$9,645.35	0.99845	$9,630.40	$9,630.40
2	FEB	$8,861.45	0.89637	$7,943.14	$17,573.54
3	MAR	$7,579.23	1.04578	$7,926.21	$25,499.74
4	APL	$8,472.39	0.99563	$8,435.37	$33,935.11
5	MAY	$9,384.74	0.98724	$9,264.99	$43,200.10
6	JUN	$9,381.36	1.00376	$9,416.63	$52,616.74

SUMMARY, TOTAL RAW AMOUNT AND AVERAGE SCALE FACTOR TO DATE: Raw Amount: $53,324.52. Scale factor: 0.98787.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
10 ' Program Name: P-USG.BAS
12 '       Purpose: Demo of BASIC PRINT USING Statement
20 '    Written By: Bernard H. Robinson, Jr
30 ' Date Written: April 30, 1986
40 '       Version: 011
100 DEFDBL D
110 DEFINT C,I,K
120 GOSUB 5000 ' Initialize
130 GOSUB 1000 ' Print heading, first method
140 GOSUB 2000 ' Compute and print table of data, first method
150 GOSUB 2400 ' Print summary data, first method
160 GOSUB 3000 ' Print heading, second method
170 GOSUB 3100 ' print data table, second method
180 GOSUB 3300 ' Print Summary, second method
190 END
1000 ' Print heading, first method
1010 C1 = 1 : C2 = 8 : C3 = 15 : C4 = 27 : C5 = 36 : C6 = 50 ' Col. Pos.
1020 CLS ' Clear screen
1030 ' Print first line of heading
1040 PRINT USING "&"; "LINE"; ' Line Number
1050 PRINT TAB(C3 + 1); ' Tab to next print position
1060 PRINT USING "&"; "RAW"; ' Raw Amount
1070 PRINT TAB(C4); ' Tab to next print position
1080 PRINT USING "&"; "SCALE"; ' Scale Factor
1090 PRINT TAB(C5+2); ' Tab to next print position
1100 PRINT USING "&"; "NET"; ' Net Amount
1110 PRINT TAB(C6+2); ' Tab to next print position
1120 PRINT USING "&"; "ACCUM." ' Accum. Amount
1130 ' Print second line of heading
1140 PRINT USING "\  \"; "NO. "; ' Line Number
1150 PRINT TAB(C2-1); ' Tab to next print position
1160 PRINT USING "\     \"; "MONTH"; ' Month
1170 PRINT TAB(C3+1); ' Tab to next print position
1180 PRINT USING "\     \"; "AMOUNT"; ' Raw Amount
1190 PRINT TAB(C4); ' Tab to next print position
1200 PRINT USING "\     \"; "FACTOR"; ' Scale Factor
1210 PRINT TAB(C5+2) ' Tab to next print position
1220 PRINT USING "&"; "AMOUNT"; ' Net Amount
1230 PRINT TAB(C6+2); ' Tab to next print position
1240 PRINT USING "&"; "AMOUNT" ' Accum. Amount
1250 ' Print third line of the heading (dashes)
1260 PRINT USING "&"; "---";
1270 PRINT TAB(C2-1);
1280 PRINT USING "&"; STRING$(5,"-");
1290 PRINT TAB(C3);
1300 PRINT USING "&"; STRING$(9,"-");
1310 PRINT TAB(C4);
1320 PRINT USING "&"; STRING$(7,"-");
1330 PRINT TAB(C5);
1340 PRINT USING "&"; STRING$(10,"-");
1350 PRINT TAB(C6);
1360 PRINT USING "&"; STRING$(11,"-")
1370 RETURN
2000 ' First Routine to compute and print numerical data table
2010 DAAMT(0)=0
2020 FOR K=1 TO 6
2030 DN(K) = DR(K)*DS(K) ' Compute net amount
2040 DA(K) = DN(K) + DA(K-1) ' Compute accumulated amount
2050 GOSUB 2200 ' Print line of numerical data
2060 NEXT K : RETURN
2200 ' First method, print line of data
2210 PRINT USING "###"; K; ' Line Number
2220 PRINT TAB(C2);
2230 PRINT USING "\ \"; MO$(K); ' Month
2240 PRINT TAB(C3);
2250 PRINT USING "$$###,.##"; DR(K); ' Raw Amount
2260 PRINT TAB(C4);
2270 PRINT USING "#.#####"; DS(K); ' Scale Factor
2280 PRINT TAB(C5);
2290 PRINT USING "$$#####,.##"; DN(K); ' Print Net Amount
2300 PRINT TAB(C6);
2310 PRINT USING "$$#####,.##"; DA(K) ' Print Accum Amount
2320 RETURN
2400 ' First method, print summary data
2410 DRT = 0 : DST = 0
2420 FOR I = 1 TO 6 : DRT = DRT + DR(I)
2430 DST = DST + DS(I) : NEXT I
2440 DSA = DST/6#
2450 PRINT "SUMMARY, TOTAL RAW AMOUNT AND AVERAGE SCALE FACTOR TO DATE:"
2460 PRINT USING "Raw Amount: $$#####,.##_."; DRT;
2470 PRINT TAB(27);
2480 PRINT USING "Scale Factor: #.#####_."; DSA
2490 RETURN
3000 ' Print heading, second method
3010 PRINT "LINE          RAW          SCALE       NET            ACCUM."
3020 PRINT "NO.   MONTH   AMOUNT       FACTOR      AMOUNT         AMOUNT"
3030 PRINT "---   -----   ----------   --------   -----------    -----------"
3040 RETURN
3100 ' Second routine for printing data
3110 FOR K=1 TO 6 : GOSUB 3200 : NEXT K
3120 RETURN
3200 ' Print line of data, second method
3210 PRINT USING "###    \ \     $$###,.##    #.#####  ";K;MO$(K);DR(K);DS(K);
3220 PRINT USING "$$#####,.##     $$#####,.##";DN(K);DA(K)
3230 RETURN
3300 ' Print summary, second method
3310 PRINT "SUMMARY, TOTAL RAW AMOUNT AND AVERAGE SCALE FACTOR TO DATE:"
3320 PRINT USING "Raw Amount: $$#####,.##_.  Scale factor: #.#####_.";DRT;DSA
3330 RETURN
5000 ' Initialize
5010 DIM DR(6), DS(6), DN(6), DA(6), MO$(6)
5020 DATA 9645.35#, 8861.45#, 7579.23#, 8472.39#, 9384.74#, 9381.36#
5030 FOR I=1 TO 6 : READ DR(I) : NEXT I
5040 DATA 0.99845#, 0.89637#, 1.04578#, 0.99563#, 0.98724#, 1.00376#
5050 FOR I=1 TO 6 : READ DS(I) : NEXT I
5060 DATA "JAN","FEB","MAR","APL","MAY","JUN"
5070 FOR I=1 TO 6 : READ MO$(I) : NEXT I
5080 RETURN
```


Turbo Exchange

The Turbo Pascal™ Software Information eXchange

Megabytes of Turbo Pascal code, including many commercial quality programs, are now available from a single source for the low price of

$6 per diskette*

*non-member price is $7.50 per disk

Each diskette is packed with programs and/or routines organized to offer solutions in a selected area of interest. You will save countless hours and learn from the work of your fellow Turbo programmers.

Some of the selections currently available are:

106. DOS Functions & Interrupts
504. Telecommunications Programs
402. Business Financial Analysis
203. Prog. Utilities, Xref, etc.
206. Source Code Debugging
901. Animation, Sprites w/Graphics
603. Turbo Games
212. Screen Design Tools
201. Windowing & Menu Tools
903. Custom Graphics Font Creation

Many more selections available with new titles added regularly. For a complete list send for the TURBO S.I.X. catalog, or better yet join TURBO S.I.X As a member you will:

- Pay only **$6** per disk. Non-members pay **$7.50.**
- Get a MONTHLY newsletter covering what's new in Turbo software, programming tips and other topics of interest to the Turbo Pascal user. Edited by Steve Wood, author of Using Turbo Pascal. (Osborne/McGraw-Hill)
- Get the Turbo S.I.X. catalog and regular updates FREE.

Your one year membership is ONLY $19.95

The Turbo S.I.X. catalog is available to non-members for $2.95. Non-Members may purchase diskettes for $7.50 each.

Overseas orders add $12.00 for 1 year membership and $1.50 per disk for Air Mail shipping.

Texas residents add 5% sales tax.

(800) 628-2828 Ext. 606
TURBO S.I.X.
P.O. Box 8373, Dept. P1
Waco, Texas 76710
3101 Mitchell, Waco, TX 76708 • (817) 753-2182

Circle Number **61** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

VINTAGE

BACK ISSUES
1983 ~ 1986

•Vol. 1, No. 1: Norton batch files and Pascal I/O; Program style manual; COBOL on the IBM PC; Benchmarks; System analysis; BASIC modules; Rounding error.

•Vol. 1, No. 2: Defining character sets; C86 BIOS calls; Troubleshooting; Copy protection; Pascal code; Input routines; Writing an assembler.

•Vol. 1, No. 3: ASM dumb terminal; ASM tutorial routines; C source code; ASM graphics line drawing; Text editors reviewed; Utilities; Self-documenting programs; Text files to dBASE II; dBASE II PEEKS & POKES.

•Vol. 1, No. 4: Diskette directory via Pascal; MASM WHILEWEND; C/ASM interface; ASM stay resident routine; Structured programming; Software law; Pascal tutorial; Default values and variable names.

•Vol. 1, No. 5: *Multi-tasking software review*; Transporting C programs; Programmer's calculator; dBASE II bargraphs; Utilities; ROM BIOS equates; Pascal check program; *Combining* modules in Assembler; Structured programming in BASIC; Agreement outline.

•VOL. 1, No. 6: ASM printer configuration; DOS patch utility; Hi-speed graphics; Software ownership; Structured programming revisited; C and BASIC compared; Software protection; dWINDOW review.

•Vol. 2, No. 1: Freelance consulting; Sorting in C; Pascal text editor source code; CRT distress; Compiled BASIC; Assembly modules for C; Designing a relational database; Ultra Utilities; Punctuation and Style reviewed.

•Vol. 2, No. 2: *Interview with Brian Kernighan*; Understanding C through assembly language; Utility routines in C and assembly; Music in C; Automatic line number optimizing in BASIC; Assembly language tutorial; Full-screen input routines in BASIC; dBASE II printer control codes; Pascal tools reviewed; C programming support products.

•Vol. 2, No. 3: Software development environments; Creating a productive screen handling tool; Assembly tutorial; Generating graphics within windows; Composition and editing system for musical ideas; Hybridizing assembly and BASIC; Creating screens; Linking Assembler to Pascal and BASIC.

•Vol. 2, No. 4: Writing a custom ROM BIOS for PC DOS; RISC, Transputers, and mass storage; The future of disk protection; 8088 assembly macros; Random access files in BASIC; C utility routines you can buy; Using the PMATE macro language; Choosing a database manager; E/SP syntax-checking; Pascal editor reviewed.

•Vol. 2, No. 5: Developing the APX CORE Executive; From assembler to C via a 211 byte .EXE main(); Scroll control using BASIC and Assembler; Faster screen handling with BASIC and ASM; Random access files, Part 2; RTOOL, a PC-resident programmer's toolkit; dBkey reviewed; Pascal's pointer power.

•Vol. 2, No. 6: Periscope—building a hardware board for the IBM PC; Intel 80286 bugs by FNE; AT&T surprises all; Field Usage and pointer subtraction in C; Calendar functions for BASIC; Memory segmentation; Reduce your .EXE file sizes; Enhancing Debug; dBASE II to dBASE III? THINKTANK reviewed.

•Vol. 3, No. 1: November's Comdex; C—its uses and abuses; Using the C preprocessor; Memory segmentation examples; Index files, Part 1; Structured programming language; Periscope, a symbolic debugger.

•Vol. 3, No. 2: UNIX! (again); Artificial Intelligence; Doctor C's tips—pointers to functions; Calendar date routines in BASIC; Calculate record layouts, automatically!

•Vol. 3, No. 3: NAPLPS and the professional programmer; Instant-C; Printer control from a batch file; Repositioning the stars in Starfinder; Reducing EXE file size through data positioning; tools for better programming.

•Vol 3, No. 4: More UNIX stuff; C programming books, Part 1; C code excerpts, Part 1; How to cure the "dreaded locked keyboard" condition; Calling MS DOS from Turbo Pascal; DOS 2.x sort and find filters; the SET Tools.

•Vol 3, No. 5: Windows done here; C programming books, Part 2; C code excerpts, Part 2; A menu-driven PC using DISPLAY.COM; Turbo Pascal and MASM: making them work together; dBASE II routines; Turbo Pascal 3.0; TopView Programmer's Toolkit.

•Vol. 3, No. 6: About C's HLL overhead; How to recover all your costs; Report on the Comdex and Usenix shows; The Santa Cruz Operation XENIX for the IBM PC; Common C bugs; CTOOLS2; Turbo Pascal 3.0 and subdirectories; More dBASE II routines; Clipper reviewed; A better way to interface BASIC and Assembler.

•Vol. 4, No. 1: FNE on PC/AT hard disk problems and plunging 64K DRAM prices; AUTOINC explained; Sorting using an AVL tree; Combining binary trees; Using DOS to call programs from Turbo Pascal; An operating system for the people; Final installment of Brett Salter's dTOOLS; Two microcomputer graphics books reviewed.

•Vol. 4, No. 2: FNE on Intel 80386, single- vs. multi-user, multi-tasking computing, benchmarks; The home office syndrome; Frank Greco's industry news; Funky string functions; Capturing file handles in Turbo Pascal; Proposed ANSI C Standard, Part 1; Optimizing for speed in Assembler; Memory resident control over print screen functions; Structuring BASIC output with field statements; QuickBASIC Compiler and networking; Programming for Printer Output reviewed.

•Vol. 4, No. 3: FNE—another RISC machine, the paging game, software viewpoint, programmer talent, and Aloha, 52032; Sun Microsystems' next generation PC; Consulting—reusable modules; INT24.PAS, a critical error handler for Turbo Pascal; Memory model independence in Assembler; An approach to windowing for the assembly language programmer; Proposed ANSI C Standard, Part 2; Microsoft and Lattice C 3.0 reviewed; Accelerator boards for software development reviewed.

USE THE CARD TO ORDER BACK ISSUES. IF IT HAS ALREADY BEEN USED, YOU CAN ORDER BY PHONE - 503-484-2162

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Structure of dBASE II and III File Headers

by Miriam Liskin
Editor, Computer Currents

A dBASE file can be corrupted so that the record count in the header does not match the actual number of records in the file.

If you understand the structure of the dBASE file header, you can patch the record count directly with DDT (on CP/M systems), DEBUG (under MS DOS), or a disk patching utility.

If you understand the structure of the header, you can also change field names, and, in some circumstances, field types instead of having to go through the time-consuming process of copying the data base to a delimited or SDF text file, changing the field names, and then appending the records back from the text file. And if you are a programmer trying to access dBASE from BASIC, Pascal, C, or some other language, an understanding of the structure is essential to writing programs to read .DBF files.

Patching with DDT and Debug

DDT and DEBUG can operate only on files which fit entirely in RAM memory. Typically, in a 64K CP/M system, once you have loaded the operating system and DDT into memory, you have room left for a data base somewhere between 30K and 40K in size. Under MS DOS, you can estimate the maximum file size by subtracting from your system's total RAM memory the size of the operating system (about 13K for DOS 1.1 or 40K for DOS 2.0 or 2.1) and 12K for DEBUG itself. In order to patch larger data bases, you must use a disk patching utility such as the public domain DU.COM for 8-bit CP/M computers, or the Norton Utilities for MS DOS systems.

dBASE II File Headers

Figure 1 lists the structure of a typical mailing list data base, and Figure 2 shows the first 256 bytes of the dBASE II file header as displayed by DEBUG.

In a dBASE II data base, the header is 521 bytes long. The file is identified as a dBASE II data base by the first byte (location 100H in the DEBUG display), which must be 02. The next two bytes contain the all-important record count. The low order byte comes first, and then the high order byte. Debug displays this number in hexadecimal notation.

If you are not conversant with base 16 arithmetic and don't have a calculator or conversion table handy, you can simply patch in some larger number which you do know how to express in hex, and then PACK or COPY your file. Beginning with a very high record count insures that dBASE will in fact read all of the records physically present on the disk, because both commands count records as they go. If you use PACK, your original file will end up with the correct record count; if you COPY the file instead, the original should be erased afterward and the copy renamed to become your working copy of the file. Entering FF and FF into these two bytes is therefore always a safe choice — this tells dBASE that your data base contains FFFF, or 65535 (i.e., 64K) records — the maximum permissible number.

The next three bytes contain the month, day, and year of last update, as hex bytes. In the sample file in Figure 2, these three bytes are 07 (month 7), 10 (day 16), and 55 (year 85), indicating that the file was last updated on July 16, 1985. The next two bytes contain the total record length as a 16-bit number (like the record count, the order of these two must be reversed to form a four-digit hexadecimal number). In our example, the record length is 00AC, or (10 x 16) + 12 = 172. Be careful not to change this number — if it does not match the actual length of the records in the file, dBASE will not be able to read your data base correctly.

Next come the field names, types and lengths. Each field description takes up 16 bytes, and space is allocated for all 32 fields permitted by dBASE II, so this portion of the header is 16 x 32 = 512 bytes long. In a data base which contains fewer than 32 fields, the unused locations are filled with hex 0's. In each group of 16 bytes, the first ten contain the field name, as ordinary ASCII characters, except that if a field name is shorter than 10 characters, the extra spaces are filled with hex 00's, not, as you might expect, spaces. The eleventh byte is always 00. The twelfth byte contains the field type, as an ASCII character ("C", "N", or "L"), and the thirteenth is the field length. The next two bytes form a 16-bit number expressing the offset of the field from the beginning of the record. Finally, the last byte contains the number of decimal places for numeric fields.

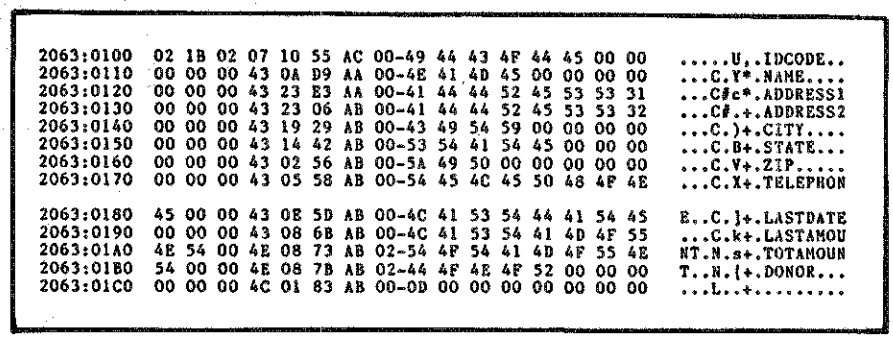
The 521st byte of the header contains a <RETURN> (0D hex) if all 32 fields are in use, or a 00 if your structure contains fewer. Following the header, you will find the data records themselves, stored as fixed length ASCII records, with each field filled out to its maximum length with blank spaces and no separators between fields or between records. Before the first field begins, each record contains an asterisk ("*") if it is marked for deletion, or a blank space if not.

Although you will need all of this information if you are writing programs to access a dBASE data base, the only things you can safely patch are the record count, the date of last update (although this can be more easily changed through dBASE by editing any record in the file), and the field names or types. Adding or deleting a field, changing the length of a field, or altering the total record length will result in a header that does not match the data records that follow. You must also be careful not to enter any punctuation other than a colon (":") into a field name, or to create duplicate field names. If you change field types, you must insure that the new type is appropriate for the contents of the field, and you must not specify a field type other than "C", "N", or "L".

```
STRUCTURE FOR FILE:  C:MAILLIST.DBF
NUMBER OF RECORDS:   00539
DATE OF LAST UPDATE: 07/16/85
PRIMARY USE DATABASE
FLD       NAME       TYPE WIDTH    DEC
001       IDCODE     C    010
002       NAME       C    035
003       ADDRESS1   C    035
004       ADDRESS2   C    025
005       CITY       C    020
006       STATE      C    002
007       ZIP        C    005
008       TELEPHONE  C    014
009       LASTDATE   C    008
010       LASTAMOUNT N    008      002
011       TOTAMOUNT  N    008      002
012       DONOR      L    001
** TOTAL **                00172
```

Figure 1:

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



```
2063:0100  02 1B 02 07 10 55 AC 00-49 44 43 4F 44 45 00 00   .....U,.IDCODE..
2063:0110  00 00 00 43 0A D9 AA 00-4E 41 4D 45 00 00 00 00   ...C.Y*.NAME....
2063:0120  00 00 00 43 23 E3 AA 00-41 44 44 52 45 53 53 31   ...C#c*.ADDRESS1
2063:0130  00 00 00 43 23 06 AB 00-41 44 44 52 45 53 53 32   ...C#.+.ADDRESS2
2063:0140  00 00 00 43 19 29 AB 00-43 49 54 59 00 00 00 00   ...C.)+.CITY....
2063:0150  00 00 00 43 14 42 AB 00-53 54 41 54 45 00 00 00   ...C.B+.STATE...
2063:0160  00 00 00 43 02 56 AB 00-5A 49 50 00 00 00 00 00   ...C.V+.ZIP.....
2063:0170  00 00 00 43 05 58 AB 00-54 45 4C 45 50 48 4F 4E   ...C.X+.TELEPHON

2063:0180  45 00 00 43 0E 5D AB 00-4C 41 53 54 44 41 54 45   E..C.]+.LASTDATE
2063:0190  00 00 00 43 08 6B AB 00-4C 41 53 54 41 4D 4F 55   ...C.k+.LASTAMOU
2063:01A0  4E 54 00 4E 08 73 AB 02-54 4F 54 41 4D 4F 55 4E   NT.N.s+.TOTAMOUN
2063:01B0  54 00 00 4E 08 7B AB 02-44 4F 4E 4F 52 00 00 00   T..N.{+.DONOR...
2063:01C0  00 00 00 4C 01 83 AB 00-0D 00 00 00 00 00 00 00   ...L..+.........
```

Figure 2:

Instant-C™:
The *Fastest* Interpreter for C

Runs your programs 50 to 500 times faster than any other C language interpreter.

Any C interpreter can save you compile and link time when developing your programs. But only *Instant-C* saves your time by running your program at compiled-code speed.

Fastest Development. A program that runs in one second when compiled with an optimizing compiler runs in two or three seconds with *Instant-C*. Other interpreters will run the same program in two minutes. Or even ten minutes. Don't trade slow compiling and linking for slow testing and debugging. *Only Instant-C will let you edit, test, and debug at the fastest possible speeds.*

Fastest Testing. *Instant-C* immediately executes any C expression, statement, or function call, and display the results. Learn C, or test your programs faster than ever before.

Fastest Debugging. *Instant-C* gives you the best source-level debugger for C. Single-step by source statement, or set any number of conditional breakpoints throughout your program. Errors always show the source statements involved. Errors always show the source statements involved. Once you find the problem, test the correction in seconds.

Fastest Programming. *Instant-C* can directly generate executable files, supports full K & R standard C, comes with complete library source, and works under PC-DOS, MS-DOS, or CP/M-86. *Instant-C gives you working, well-tested programs faster than any other programming tool.* Satisfaction guaranteed, or your money back in first 31 days. *Instant-C* is $495.

Rational
Systems, Inc.

P.O. Box 480
Natick, MA 01760
(617) 653-6194

dBASE III Headers

Although dBASE III does allow you to change field names using MODIFY STRUCTURE, it accomplishes the change the same way you would at command level in dBASE II — by creating a temporary copy of the structure to make the changes, and then appending the records back into the new structure — a much slower process than patching the structure with DEBUG. Also, since the dBASE III PACK command cannot restore the correct record count if a data base is damaged by failure to close the file before exiting from the program, it is if anything more important than with dBASE II to be able to patch in the right record count to repair a corrupted data base. Be aware, however, that since dBASE III takes advantage of all the RAM memory in your system, you can append many more records to a file without actually writing any data to the disk. If the computer is turned off in the middle of a data entry session, any records still in the memory buffer at the time cannot be retrieved by any means — and there are likely to be more of them than with the equivalent dBASE II file.

Figure 3 is Figure 1's mailing list written in dBASE III. Figure 4 is the first 512 bytes of the file as displayed by DEBUG.

As with dBASE II, the first byte of the header identifies the file as a dBASE III data base. This byte is 83 for a data base which contains memo fields and therefore has an associated .DBT file (which contains the text of the memo fields). Otherwise, the first byte is 03. The next three bytes contain the date of last update in year, month, day order. Since the sample file in Figure 4 was created on July 30, 1985, these three bytes are 55 (year 85), 07 (month 7), and 1E (day 30).

The next four bytes contain the record count, as a thirty-two bit number. As with dBASE II, these bytes are arranged from least to most significant.

The largest number that can be expressed by two bytes is FFFF, which is 65,535 (64K) decimal — the maximum number of records in a dBASE II data base. The maximum size of a four byte number is hex FFFFFFFF, or 4,294,967,295 decimal, which exceeds dBASE III's limit of 1 billion records in a file. In our example, these four bytes are 1B, 02, 00, and 00. Thus, the record count is 021B hex, or (2 x 256) + (1 x 16) + 11 = 539 decimal. As mentioned above, you can always patch in a larger number and then PACK or COPY the file to yield a data base with the correct count. Don't use FFFFFFFF, however — you will lose all or most of your data (not surprising since this is more than the max-

Circle Number 20 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

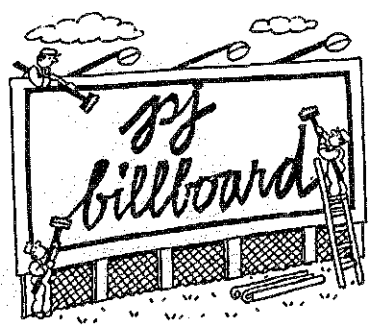
imum number of allowable records).

The next two bytes specify the length of the file header, as a 16-bit number. Unlike dBASE II, dBASE III does not use a fixed length record header: You may have a maximum of 128 fields, and space in the header is allocated only for the number of fields you have actually defined. Each field, as we will see shortly, occupies 32 bytes. In the sample file, the length of the header is 1A2 (hex), or 418 (decimal) bytes: 32 bytes for the fixed portion of the header we are describing now plus 32 x 12 = 384 bytes for the 12 fields in the data base, followed by 1 byte for the RETURN character (decimal ASCII 13) and one for the hex 0 used to signal the end of the header.

The next two bytes contain the total record length as a 16-bit number, just as in the dBASE II file header. In our example, the record length is 00AC, or (10 x 16) + 12 = 172. Remember that you must not change this number — if it does not match the actual length of the records in the file, dBASE will not be able to read your data base correctly. The next 20 bytes are described by Ashton-Tate as "reserved" — in the sample data base, they are all hex 0's.

Next come the field names, types and lengths. In each group of 32 bytes, the first ten contain the field name, as ordinary ASCII characters, with field names shorter than 10 characters filled out with hex 00's. The eleventh byte is always 0. The twelfth byte contains the field type, as an ASCII character ("C", "N", "L", "D", or "M"). The next four bytes form a 32-bit number expressing the address in memory of the field — the offset of the field from the beginning of the record. The next two bytes contain the field length, and, for numeric fields, the number of decimal places. The last 14 bytes in each group are "reserved."

Just as with dBASE II, the header is followed by the data records themselves, stored as fixed length ASCII records, with each field filled out to its maximum length



Brand New From Peter Norton

A PROGRAMMER'S EDITOR

that's *lightning fast* with the *hot* features programmers need

A WONDERFUL EDITOR FOR PROGRAMMERS FROM PETER NORTON

THE NORTON EDITOR™

only $50

Easily customized and saved ▲ *Split-screen editing* ▲ *A wonderful condensed/outline display* ▲ *Structured programming features:* ● *Auto-indenting for* Pascal *and* C ● *A hot "find matching punctuation" feature* ▲ *Great word features too:* ● *Paragraph reformat* ● *Word wrap, word action* ● *You can write your programs* and *your manuals with this one editor* ▲ *Mouse support, too* ▲ *Great for* assembler, *fabulous for* Pascal *and* C

A STEAL AT $50!

Direct from the man who gave you *The Norton Utilities, Inside the IBM PC,* and the *Peter Norton Programmer's Guide.*

"This is the programmer's editor that I wished I'd had when I wrote my *Norton Utilities.* Now I've got it for you to use. You can *program your way to glory* with *The Norton Editor.*"

Peter Norton

Peter Norton Computing, Inc., 2210 Wilshire Boulevard, Santa Monica, CA 90403, 213-453-2361. Visa, Mastercard and phone orders welcome.

The Norton Editor™ is a trademark of Peter Norton Computing, Inc. © 1986 Peter Norton Computing.

Circle Number 17 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C & PASCAL PROGRAMMERS

Blaise Computing provides a broad range of programming tools for Pascal and C programmers, with libraries designed for serious software development. You get carefully crafted code that can be easily modified to grow with your changing needs. Our packages are shipped complete with comprehensive manuals, sample programs and source code.

C TOOLS PLUS
$175.00

NEW! Full spectrum of general-purpose utility functions; windows that can be stacked, removed, and accept user input; interrupt service routines for resident applications; screen handling including EGA 43-line text mode support and direct screen access; string functions; and DOS file handling.

PASCAL TOOLS/TOOLS 2
$175.00

Expanded string and screen handling; graphics routines; easy creation of program interfaces; memory management; general program control; and DOS file support.

VIEW MANAGER
$275.00

Complete screen management; paint data entry screens; screens can be managed by your application program; block mode data entry or field-by-field control. Specify C or IBM/MS-Pascal.

ASYNCH MANAGER
$175.00

Full featured asynchronous communications library providing interrupt driven support for the COM ports; I/O buffers up to 64K; XON/XOFF protocol; baud rates up to 9600; modem control and XMODEM file transfer. Specify C or IBM/MS-Pascal.

Turbo POWER TOOLS PLUS
$99.95

NEW! Expanded string support; extended screen and window management including EGA support; pop-up menus; memory management; execute any program from within Turbo Pascal; interrupt service routine support allowing you to write memory resident programs; schedulable intervention code.

Turbo ASYNCH PLUS
$99.95

Complete asynchronous communications library providing interrupt driven support for the COM ports; I/O buffers up to 64K; XON/XOFF protocol; and baud rates up to 9600.

RUNOFF
$49.95

NEW! Text formatter written especially for programmers; flexible printer control; user-defined variables; index generation; and general macro facility. Crafted in Turbo Pascal.

EXEC
$95.00

Program chaining executive. Chain one program from another even if the programs are in different languages. Shared data areas can be specified.

ORDER TOLL-FREE 800-227-8087!

BLAISE COMPUTING INC.

2560 Ninth Street, Suite 316 Berkeley, CA 94710 (415) 540-5441

Circle Number 2 on the Reader Service Card

with blank spaces, and no separators between fields or between records. Before the first field begins, each record contains an asterisk ("*") if it is marked for deletion, or a blank space if not.

The only things you can safely patch are the record count, the date of last update (although this can be more easily changed through dBASE by editing any record in the file), and the field names or types. You must be careful not to enter any punctuation other than an underscore ("_") into a field name, or to inadvertently create duplicate field names. If you change field types, you must insure that the new type is appropriate for the contents of the field, and you must not specify a field type other than the five allowed in d BASE III:"C", "N", "L", "D", and "M". □

Figure 3:

```
Structure for database : C:maillist.dbf
Number of data records :      539
Date of last update    : 07/30/85
Field  Field name  Type       Width     Dec
    1  IDCODE      Character     10
    2  NAME        Character     35
    3  ADDRESS1    Character     35
    4  ADDRESS2    Character     25
    5  CITY        Character     20
    6  STATE       Character      2
    7  ZIP         Character      5
    8  TELEPHONE   Character     14
    9  LASTDATE    Character      8
   10  LASTAMOUNT  Numeric        8       2
   11  TOTAMOUNT   Numeric        8       2
   12  DONOR       Logical        1
** Total **                     172
```

Figure 4:

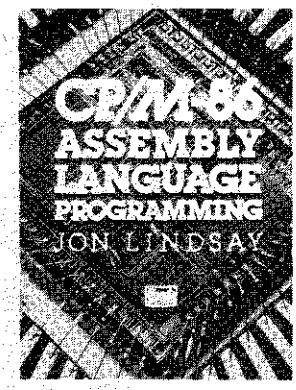
```
61D6:0100  03 55 07 1E 1B 02 00 00-A2 01 AC 00 00 00 00 00   .U.....".......
61D6:0110  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
61D6:0120  49 44 43 4F 44 45 00 00-00 00 00 43 0F 00 97 4E   IDCODE.....C...N
61D6:0130  0A 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
61D6:0140  4E 41 4D 45 00 00 00 00-00 00 00 43 19 00 97 4E   NAME.......C...N
61D6:0150  23 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   #...............
61D6:0160  41 44 44 52 45 53 53 31-00 00 00 43 3C 00 97 4E   ADDRESS1...C<..N
61D6:0170  23 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   #...............

61D6:0180  41 44 44 52 45 53 53 32-00 00 00 43 5F 00 97 4E   ADDRESS2...C_..N
61D6:0190  19 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
61D6:01A0  43 49 54 59 00 00 00 00-00 00 00 43 78 00 97 4E   CITY.......Cx..N
61D6:01B0  14 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
61D6:01C0  53 54 41 54 45 00 00 00-00 00 00 43 8C 00 97 4E   STATE......C...N
61D6:01D0  02 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
61D6:01E0  5A 49 50 00 00 00 00 00-00 00 00 43 8E 00 97 4E   ZIP........C...N
61D6:01F0  05 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................

61D6:0200  54 45 4C 45 50 48 4F 4E-45 00 00 43 93 00 97 4E   TELEPHONE..C...N
61D6:0210  0E 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
61D6:0220  4C 41 53 54 44 41 54 45-00 00 00 44 A1 00 97 4E   LASTDATE...D!..N
61D6:0230  08 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
61D6:0240  4C 41 53 54 41 4D 4F 55-4E 54 00 4E A9 00 97 4E   LASTAMOUNT.N)..N
61D6:0250  08 02 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
61D6:0260  54 4F 54 41 4D 4F 55 4E-54 00 00 4E B1 00 97 4E   TOTAMOUNT..N1..N
61D6:0270  08 02 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................

61D6:0280  44 4F 4E 4F 52 00 00 00-00 00 00 4C B9 00 97 4E   DONOR......L9..N
61D6:0290  01 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
61D6:02A0  0D 00 20 4C 49 53 4B 49-4E 20 20 20 20 4D 69 72   .. LISKIN    Mir
61D6:02B0  69 61 6D 20 4C 69 73 6B-69 6E 20 20 20 20 20 20   iam Liskin
61D6:02C0  20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
61D6:02D0  32 30 30 30 20 43 65 6E-74 65 72 20 53 74 72 65   2000 Center Stre
61D6:02E0  65 74 20 20 20 20 20 20-20 20 20 20 20 20 20 20   et
61D6:02F0  20 20 20 23 31 32 32 39-20 20 20 20 20 20 20 20      #1229
```

This material first appeared in Computer Currents.

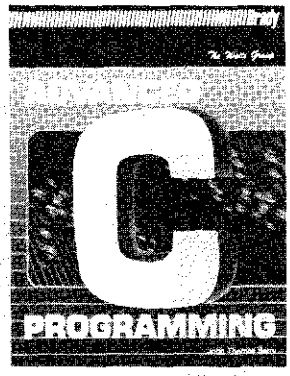
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

BRADY Speaks Your Language

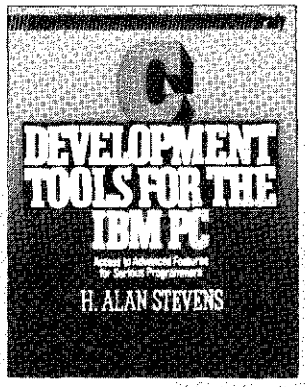
We can help you buff up your BASIC...Put a finish on your FORTH... Conquer C...Succeed with Assembler...And more! Just call toll-free or use the coupon to order today!



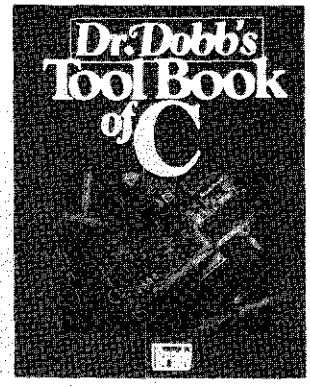
1. The perfect intro to CP/M-86 even if you've never programmed in assembly language before. Includes sections on sequential file handling and memory manipulation, and more. $21.95



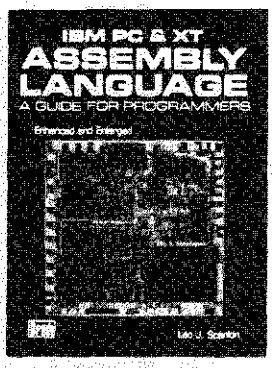
2. Beyond the basics, this guide explores new structured concepts to analyze and solve problems in C with tools of modularity, input and output functions, arrays and structures, and bit manipulation. $21.95



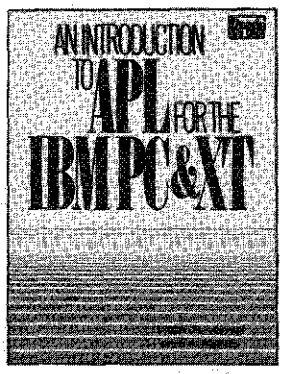
3. The special focus is on screen management and data retrieval. You'll find a library of reusable C software tools to help create interactive systems, including file management, screen forms, etc. $21.95



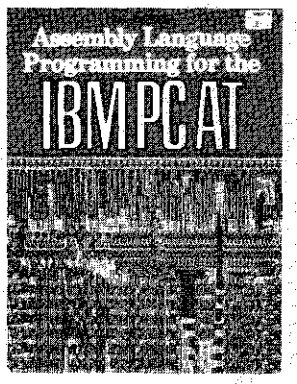
4. The best C articles from the highly respected *Dr. Dobb's Journal* dealing exclusively with C language and programming techniques. $24.95



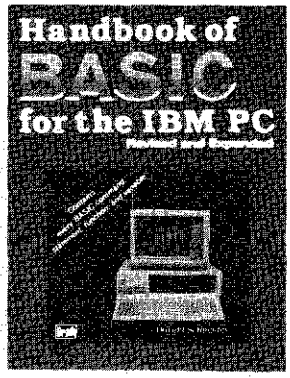
5. Our best-selling assembler book has been made even better! It now includes 30 assembler Macros and version 2.0 of the IBM Assembler. $21.95



6. For everyone who thought APL was too unorthodox to bother learning, this guide will unlock the secrets to its logic and show you how to write the fastest, most compact code. $22.95



7. The author of our best-selling assembler books now demonstrates his detailed and accurate style on the 80286 chip. $21.95



8. PC magazine calls it: "A truly remarkable book...A treasure trove of useful programming information on the IBM PC." $22.95

Now at your book or computer store.
Or order toll-free today: **800-624-0023**
In New Jersey: 800-624-0024

BRADY COMMUNICATIONS COMPANY, INC.
c/o Prentice Hall
P.O. Box 512, W. Nyack, NY 10994

Circle the numbers of the titles you want below. (Payment must be enclosed; or, use your charge card.) Add $1.50 for postage and handling.
Enclosed is check for $_____ or charge to
☐ MasterCard ☐ VISA

Acc't # _____ Exp. date _____
Signature _____
Name _____
Address _____
City _____ State _____ Zip _____
(New Jersey residents, please add applicable sales tax.)
Dept. 3

GR-BLPJ-CB(6)

1 (0-89303-390-1) 2 (0-89303-473-8) 3 (0-89303-612-9) 4 (0-89303-599-8)
5 (0-89303-575-0) 6 (0-89303-567-X) 7 (0-89303-484-3) 8 (0-89303-510-6)

Brady

Circle Number 37 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PORTABLE POWER WITH MasterFORTH

Whether you program on the **Macintosh**, the **IBM PC**, an **Apple II** series, a **CP/M** system, or the **Commodore 64**, your program will run unchanged on all the rest.

If you write for yourself, MasterFORTH will protect your investment. If you write for others, it will expand your marketplace.

Forth is interactive – you have immediate feedback as you program, every step of the way. Forth is fast, too, and you can use its built-in assembler to make it even faster. MasterFORTH's relocatable utilities and headerless code let you pack a lot more program into your memory. The resident debugger lets you decompile, breakpoint and trace your way through most programming problems. A string package, file interface and full screen editor are all standard features. And the optional target compiler lets you optimize your application for virtually any programming environment.

The package exactly matches *Mastering Forth* (Brady, 1984) and meets all provisions of the Forth-83 Standard.

MasterFORTH standard package	$125
(Commodore 64 with graphics)	$100
Extensions	
Floating Point	$60
Graphics (selected systems)	$60
Module relocator (with utility sources)	$60
TAGS (Target Applic. Generation System) – MasterFORTH, target compiler and relocator	$495
Publications & Application Models	
Printed source listings (each)	$35
Forth-83 International Standard	$15
Model Library, Volumes 1-3 (each)	$40

(213) 821-4340

MICROMOTION

8726 S. Sepulveda Bl., #A171
Los Angeles, CA 90045

Circle Number 13 on the Reader Service Card

Jump Start

Dair Computer Systems
3440 Kenneth Drive
Palo Alto, CA 94303
(415) 494-7081

This little board, which plugs into a motherboard ROM socket and doesn't take up a slot, circumvents the memory check in the PC boot-up ROM so that the time for a cold start for a 512K PC is cut from 53 seconds to as little as 2 seconds. The contents of memory and of RAM disks can also be preserved during a warm boot.

I use the no-memory-testing boot. My PC boots from a hard disk, which can't get up to speed before the fast boot routine accesses the disk, so the first try at a cold start with *Jump Start* is not successful. However, I just listen until the hard disk reaches operating speed, do a warm start, and I'm booted almost instantly. A boon when you lock the machine as much as I do.

The only worry I have is that the first unsuccessful boot will produce a crippled operating system that will take off across my hard disk like Godzilla. Nothing yet. I suppose the odds are small.

Booting from a floppy is simple—no drive has to get up to speed.

ADVERTISER INDEX

CIRCLE NUMBER		PAGE NUMBER
52	Aldebaran	5
53	Allen, Emerson & Franklin	41
54	Apex Resources, Inc.	53
1	ATRON	16
30	Austin Code Works	35
55	Barrington Systems	32-33
2	Blaise Computing	60
3	Borland	29
37	Brady Books	61
38	C-Ware	35
56	Compuview Products, Inc.	25
40	Creative Prog. Consultants	20
6	Data Base Decisions	28
41	Design Solutions	35
8	Ecosoft, Inc.	cov. 4
9	Entelekon	49
11	Gimpel Software	49
42	Harvard Softworks	3
57	Hawkeye Grafix	10
58	Interface Group	21-64
59	L.U.C.I.D.	13
63	Laboratory Microsystems, Inc.	11
44	Lahey Computer Systems	9
12	Landmark Software	42
62	MGlobal	cov. 3
64	Microhelp	19
13	Micromotion	62
65	Mnematics Vidiotex	43
15	Oliver Computing	39
16	Opt-Tech Data Processing	13
17	Peter Norton Computing	59
31	Quantum Information Systems	24
46	Quilt Computing	35
20	Rational Systems	58
33	Soft Advances	45
32	Software Masters	cov. 2
60	Sunset Technology	45
61	Turbo Six	55
49	Wizard Systems Software	47

To receive advertising information circle #89

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ BILLBOARD

PJ Billboard is a product and services directory containing listings for hardware and software of interest to the IBM PC programmer. The goal is to provide our readers with a wide range of information on current products and updates. PJ Billboard serves our advertisers by providing a low cost, flexible format in which they can present their products to our readers.

The standard layout is four columns, 9 picas wide, with copy set in 7 point on 8 point leading. PJ Billboard offers a 400-character product description, including your company name, address, and telephone number. Boldface and italics are available.

The listings are $65, prepaid, non-commissionable for a one-time insertion. Listings must be received typed, preferably on one of our convenient order forms. Copy deadlines are the same as the normal ad materials closing dates, which can be found on our rate card, or you may call the Programmer's Journal Advertising Department at 503-484-0602 for more information.

CPRINT™/RETAB™
CPRINT helps you *visualize* algorithms & structures instantly, creating *programmer* object code specifically for the *mind's eye*. Archival-references, flow chart (not just indentation) & source listings included. Nearly a "C" compiler itself, CPRINT provides appropriate error messages, shows structure & checks for logic errors. RETAB fixes the indentation of source files after changes in the program structure. Satisfaction guaranteed. MS DOS,™ AMIGA-DOS™ & UNIX™ versions available.

ENSCO
4330 Jana Vista St.
El Sobrante, CA 94803
(415) 222-7826

Advanced Programming Tools
The C Library — Over 140 macros, functions, and programs. $29.95
The C to DOS Interface — Over 70 routines to control DOS and the ROM-BIOS. $59.95
The FORTRAN to DOS Interface — Over 70 routines to control DOS and the ROM -BIOS. $59.95
The Turbo Pascal Programmer's Library — Over 175 essential routines for Turbo Pascal. $59.95

Please include $4.00 shipping. For more information write to:

Kris Jamsa Software, Inc.
Box 26031
Las Vegas, NV 89126

ROM EXPANSION FOR PC!
ALDIA SYSTEMS introduces **PC-PROMPAK**, a PROM/ROM expansion "half-card" for the IBM and IBM-compatible personal computers. **PC-PROMPAK** will support up to six 28-pin JEDEC compatible devices (ex: 2764, 27128, 27256, 27512, etc.). Addresses are individually selectable. **PC-PROMPAK** is priced at $145 for single units. Quantity discounts and OEM arrangements are available. MC/VISA

ALDIA SYSTEMS, Inc.
PO. Box 37634
Phoenix, Arizona 85069
(602) 866-1786

LAZY ASSEMBLE
Batch program assembles/links/prints multiple .ASM's with one command. Specify which error-free .LST files you want printed. Delete form feeds from .LST's if desired. Link only if all assemblies OK. Other utilities: link map re-arranger and case sensitive/insensitive high speed string search across editable file list containing full pathname and "*", "?" chars in file names. $49.95 (ck/mo).

Goldbrick Software
1376 Deerpark Dr. #22
Fullerton, CA 92631
(714) 524-6604

5,000 BASIC Programmers
Can't be wrong! Our customers wouldn't dream of working in BASIC without BDS. You won't either, once you try it. BASIC Development System (BDS) is **GUARANTEED** to improve your programming productivity or your **MONEY BACK.** Ask for a free reprint of the **RAVE REVIEWS** from PC World and InfoWorld. $125 ppd. MC/Visa.

BetaTool Systems
2020 S. Oneida St., Suite 207
Denver, CO 80224
(303) 793-0145

The SCREEN GENERATOR
SAVE TIME! Powerful Screen Designer and Memory Resident Screen Manager eliminates tedious screen management tasks from your program. Handles all screen display & data entry. Easy access to Screen Manager from your program. Not a Code Generator! NO Royalties, Not Copy Protected. Easy conversion from other sytems. Use with BASIC, TURBO ($59), FORTRAN, C, Realia COBOL, BetterBASIC, PL/M-86, or Assembler. Call if not listed. Price $125.

The West Chester Group
PO. Box 1304
West Chester, PA 19380
(215) 644-4206

CSL, C Scientific Library is available for either the Computer Innovations or Microsoft C compiler. Over 300 functions and 556-page manual. Superior to any other such library. Includes implementations of many advanced algorithms. (send $2.50 for complete annotated index of CSL functions). $245.00

Eigenware Technologies
13090 La Vista Drive
Saratoga, CA 95070
(408) 867-1184)

DIRECTORY EXTENDED™
BE IN COMMAND! New version of very popular command-oriented file manager! POWERFUL batch file generation—perform any task! FLEXIBLE file matching and sorting! CONTROL file attributes! TEST file integrity! MONITOR disk params; 2-col display! MUCH MORE! Send $35 + $5 p&h (in TX, add tax); money back guarantee!

ECONO-SOFT Dept J67
9200 North Plaza #1906
Austin, TX 78753

Repair PC's
AID/88 is a professional in-circuit tool used for self-mainteriance of IBM PC and XT system boards. In under 60 seconds, a code identifies the section in error. The manual guides you pin-by-pin to the faulty component/mode. Clones can be trouble-shot manually because I/O and memory may be controlled from front panel. **$2995 VISA/MC**

IBM "Bug" List
Two pages of error corrections to IBM PC and XT technical reference manuals. **$2.95 VISA/MC**

Fast Compiler
PL/D is a new high-speed language for implementing operating systems, utility programs and EPROM-based applications. Move easily from high to low level constructs. Extensive compile-time facilities. Self-compiles at 100 lines per second on even the slowest hardware. **$49.95 VISA/MC**

Boot Faster
Jump Start is a hardware assist to IBM PC1, PC2, or XT that speeds up s-l-o-w BIOS memory test. **$79.95 VISA/MC**

DAIR Computer Systems
3440 Kenneth Drive
Palo Alto, CA 94303
(415) 494-7081

ROMable CODE on PC!
PCLOCATE allows PC users to develop ROM-based software from MS-DOS "Exe" files. The user specifies the physical location of all segments. Output files are compatible with most PROM programmers. PCLOCATE supports the 8086, 8088, 80186, 80188, and 80286 processors. MC/VISA.

ALDIA SYSTEMS INCORPORATED
PO. Box 37634
Phoenix, AZ 85069
(602) 866-1786

Documentation
- Writing, editing, formatting for photocopying or typesetting
- MS DOS, Apple, AT&T 3B series UNIX
- Over thirty manuals
- Credits include
  Dave Winfield's BATTER UP! (Billboard Top Twenty)
  Action Inn Management (AT&T/3B2)
  SmartCAM (MS DOS, Apple)
  Programmer's Journal (Managing Editor)

Erin O'Connor
c/o Programmer's Journal
PO. Box 30160
Eugene, OR 97403
(503) 484-2162

SEARCH by CompuMagic
Search multiple program & text files for multiple strings; see matching lines in context: PARAgraph mode for word processor files; user-selectable Window around matched lines for any file. On-line help; easily learned wildcards; boolean operators; I/O redirection. Use paths in MS/PC-DOS. NOT copy-protected. For CP/M-80 or MS-DOS, only $30 with documentation!

CompuMagic, In.
Marcia Dresner
PO. Box 437
Severn, MD 21144
(301) 969-8068

TURBO PACKAGE *for Turbo Pascal*
- Crash the 64K barrier. Forget MODULA2.
- 1000 times faster than chains or overlays.
- Precompiler turns short calls into long calls. Run 500K of memory resident code.
- 32 bit integer math toolbox, ASM.
- Mix Tubo, TurboBCD, and Turbo-87.
- Free information.
- Turbo Package plus tools, $39.95.

CONVERSATIONAL COMPUTER SYSTEMS
5371 VERBENA RD
SAN ANTONIO, TEXAS 78240

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Finally, there's a major end-user computer event for the West Coast in the Fall.

Exhibitors like what the Personal Computer Faire delivers. New business. New opportunities. A professionally run conference and exposition. The right attendees and exhibitors. The highest return on your exhibiting investment.

Quality Attendees

No matter what the immediate business climate, exhibitors at a Personal Computer Faire know they can always count on a steady flow of show traffic. The impressive quality of this attendee traffic may well be your single most compelling reason for exhibiting.

National marketers, such as AT&T, IBM, Kaypro, Hayes, Sigma, Wang, Borland, NEC, Tallgrass and 3COM have relied on the Personal Computer Faire in San Francisco to help increase their market share, and support their local distributors, dealers and VARs in this hotbed of computer activity.

Regional marketers consider the annual Personal Computer Faire to be an ideal marketing vehicle for building sales, developing qualified leads and winning name recognition among thousands of local computer-knowledgeable prospects.

Value-added Show Benefits

- Hands-On Demonstrations of hardware and software.
- Scores of conference program sessions tailored to the needs of attendees.
- Featured industry speakers.
- Special professional seminars on the latest software.
- Attendees who are *true* prospects — knowledgeable and ready to buy.

Marketing & Promotion

Our unique marketing strategy balances direct marketing, publications, magazine and newspaper advertising, conference sessions tailored to attendees' needs, news conferences, year-round publicity and on-site seminars and hands-on workshops.

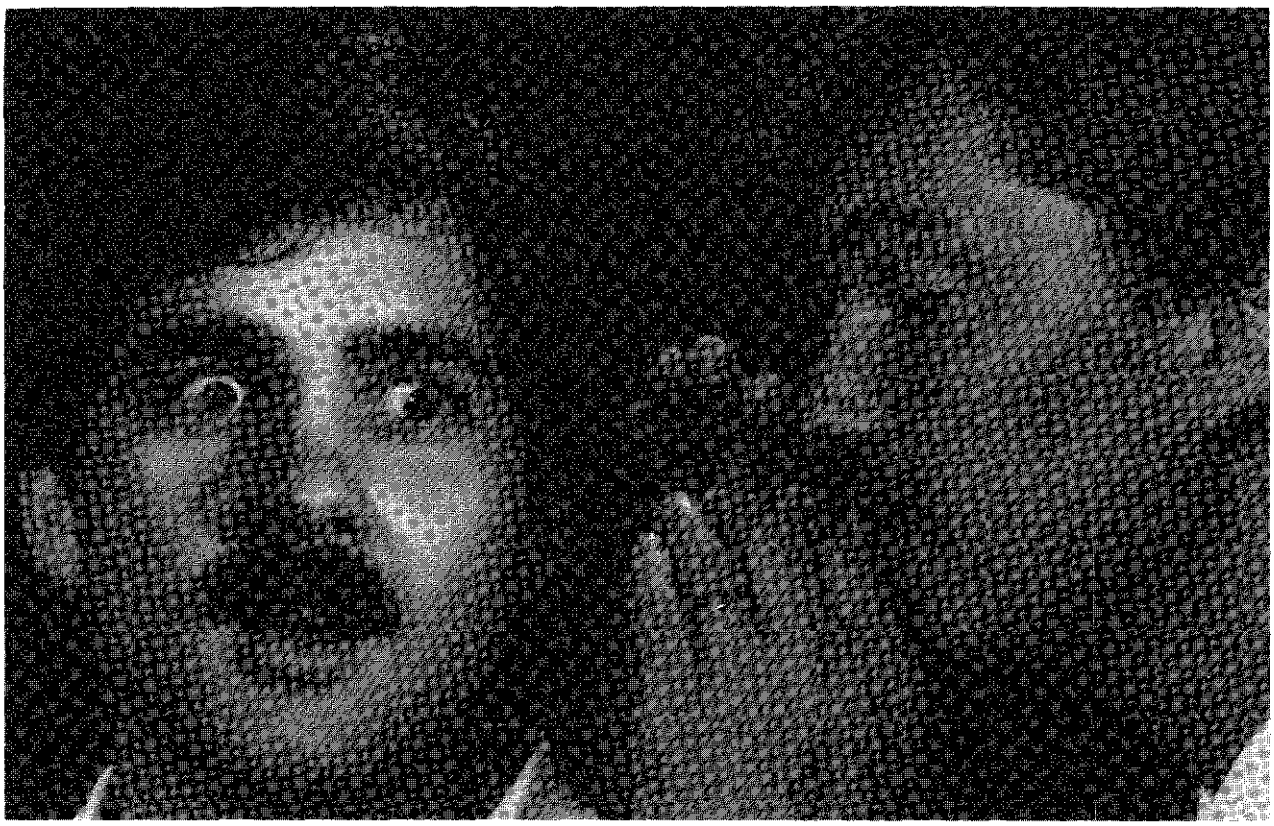
You'll experience face-to-face selling as you never have before. Our prospect-producing promo programs will deliver the best in quality and quantity. The Interface Group, Inc. produces the largest computer and communications conferences and expositions in the world. Count on us to deliver what you need to make the sale.

But take action now, before our best booth locations are committed. Contact your local Interface Group representative, or one of the offices below.

The Interface Group, Inc. · 300 First Avenue Needham, MA 02194 · (617) 449-6600
West Coast Office: 1401 Dove Street · Suite 220 · Newport Beach, CA 92660 · (714) 476-2300

Circle Number **58** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



If You Could Hear The Language They're Using!

What they're saying would astound you.

They've switched from BASIC, and PASCAL, and a host of other languages and even stand-alone database programs.

They've switched to COMP Computing Standard MUMPS...CCSM, the Database Language.

"The best software product of its kind, that I have come in contact with." *Computer Language Magazine*

"I've found the best in CCSM...fast in development and fast in execution...no data-typing problems, no concerns for program size, no concerns for file or device opens..." *R.D. Ashworth, Ph.D.*

"...5 years in Basic, Pascal, C, dBase, and Dataflex... I have never worked with a language/programming environment as responsive, easy to use and as powerful as CCSM" *P.K. Wayne, MD, Ph.D.*

CCSM, the Database Language is a powerful, flexible string-oriented language, with the features of a complete development and operating system. Typical programs are written in 1/3 to 1/5 the amount of code, compared to BASIC or PASCAL.

Multi-Tasking Available, Too!
CCSM, the Database Language offers a true multi-tasking option. Run multiple, concurrent background processes for data searches, report generation, etc.

Order CCSM, the Database Language today.

In a very short time, you'll be surprised at the language you're using.

IBM's and compatibles...128K min.

MGlobal
1601 Westheimer, Suite 201
Houston, TX 77006

1-800-257-8052
in Texas 713-529-2576

AMEX, VISA AND MC accepted by phone...order now

Single User disk and operations manual	$ 59.95
"Cookbook of MUMPS" (includes disk)	$ 24.95
SPECIAL Single user & "Cookbook"	$ 75.90
Multi-tasking	$149.95
Programmer's Toolkit I	$ 49.95
Graphics option	$ 49.95
Multi-user	$450.00
MacMUMPS (Macintosh version)	$ 89.95
shipping and handling	$ 3.00

Disks are non-copy-protected

Texas residents add 6 1/8% sales tax

IBM is a trademark of International Business Machines; Macintosh is a trademark of Apple Computer

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Circle Number 62 on the Reader Service Card

DR. JACK PURDUM

Get the proven product from the man who wrote the books on "C".

C Programming Guide

After reading the 1st edition, Jerry Pournelle (BYTE Magazine) said: "I recommend this book...Read it before trying to tackle Kernighan and Ritchie." The second edition expands this best seller and walks you through the C language in an easy-to-understand manner. Many of the error messages include references to this book making it a perfect companion to Eco-C88 for those just starting out with C.

$20

C Self-Study Guide

(Purdum, Que Corp.) Designed for those learning C on their own. The book is filled with questions-answers designed to illustrate many of the tips, traps, and techniques of the C language. Although written to complement the Guide, it may be used with any introductory text on C.

$17

C Programmer's Library

(Purdum, Leslie, Stegemoller, Que Corp.) This best seller is an intermediate text designed to teach you how to write library functions in a generalized fashion. The book covers many advanced C topics and contains many useful additions to your library including a complete ISAM file handler.

$22

CED Program Editor

CED now supports on-line function help. If you've forgotten how to use a standard library function, just type in the name of the function and CED gives you a brief summary, including function arguments. CED is a full screen editor with auto-flagging of source code errors, multiple windows, macros, and is fully configurable to suit your needs. You can edit, compile, link, and execute DOS commands from within the editor. Perfect for use with Eco-C88. For IBM PC, AT and look-alikes.

$29.95

C Source for Standard Library

Contains all of the source code for the library functions that are distributed with Eco-C88, excluding the transcendentals and functions written in assembler.

$10 ($20 if not with order)

Developer's Library

Contains all the source code for the standard library, including transcendental and assembler functions. Available with compiler purchase only.

$25 ($50 if not with order)

Ecosoft Inc.
6413 N. College Ave.
Indianapolis, IN 46220

THE FIRST PROFESSIONAL 'C' COMPILER FOR UNDER $60.00

Limited Time Offer
FREE
CED TEXT EDITOR WITH "BUILT-IN" FUNCTION HELP.

NOTHING IN THIS PRICE RANGE EVEN COMES CLOSE.

"Ecosoft's Eco-C is going to turn some major heads."

"Eco-C is the first compiler reviewed that has clearly begun implementing the coming ANSI standard. Eco-C supports prototyping."

"Eco-C performed well on all the benchmarks, generating code that was quite comparable to that of compilers 10 times as costly."

from:
Christopher Skelly
Computer Language,
Feb., 1986

"The driver program is another strength: it compiles and links a list of files and provides a simple MAKE capability."

"This compiler does handle syntax errors much better than average—no avalanche of spurious messages here."

from:
William Hunt
PC Tech Journal,
Jan., 1986

"Eco-C88 is a high-quality package... comparable to systems costing much more."

"Eco-C88 is one of the fastest..."

"It is convenient to use, works well, and produces acceptably compact and fast programs."

from:
Dr. David Clark
Byte, Jan., 1986

Minimum System Requirements:

To use Eco-C88 Release 3.0, you must have:

1. An IBM PC, XT, or AT-compatible computer with monitor.
2. 256K or more memory.
3. Two 360K disk drives, or a hard disk.
4. PC or MSDOS 2.1 or later to include the MSDOS linker.

Eco-C88, mini-make, memfiles and CED are trademarks of Ecosoft Inc.
IBM is a trademark of International Business Machines.
UNIX is a trademark of Bell Labs.
MSDOS and MASM are trademarks of Microsoft.

Full-featured C compiler. Supports all C features, data types (except bit fields), and operators.

New Language Enhancements. You also get prototyping, enum and void data types, plus structure passing and assignment.

Tiered Error Checking. All syntax errors are automatically flagged, but you can select the level of "link-like" semantic error checking you want.

Complete Standard Library. Over 200 functions, many of which are System V compatible for greater source code portability.

Screen and Memory Functions. Now you can write programs that use color, cursor addressing, even ones that let you design your own graphics functions. You also get memfiles™ that allow you to access memory outside the normal data segment as a file.

8087 and 80287 Support. If you have one of these math chips, your programs will take immediate advantage of it. If you don't have one, the code automatically switches to software floating point.

Full Screen Editor. The CED editor is a full screen editor with multiple windows, *macro commands, on-line function help,* plus a full set of editing commands. (Requires a true IBM PC compatible.) You can edit, compile, link, and execute programs from the editor which greatly reduces development time.

Includes a cc and mini-make™. The UNIX-like cc makes compiling programs a snap. You can run cc from within the CED editor.

ASM or OBJ Output. You can select assembler or relocatable output from the compiler. Both are MASM compatible and ready for use with the MSDOS linker to produce EXE files.

	Eco-C88	Lattice	Computer Inn. C86	Microsoft	Mark Williams
sieve	12	11	13	11	12
fib	43	58	46	109	—
deref	14	13	—	10	11
matrix	22	29	27	28	29

Computer Language, Feb., 1985, p. 79. Reproduced with permission.

Orders only: 1-800-952-0472

ORDER FORM CLIP & MAIL TO: Ecosoft Inc., 6413 N. College Ave., Indianapolis, IN 46220

- ☐ C Compiler $59.95 _____
- ☐ C Programming Guide $20.00 _____
- ☐ C Self-Study Guide $17.00 _____
- ☐ C Programmer's Library $22.00 _____
- ☐ CED Program Editor $29.95 _____
- ☐ C Source for Standard Library $10.00 ($20.00 if not with order) _____
- ☐ Developer's Library $25.00 ($50.00 if not with order) _____

SHIPPING _____ $4.00

TOTAL (IND. RES. ADD 5% TAX) _____

PAYMENT: ☐ VISA ☐ MC ☐ AE ☐ CHECK

CARD # _____ EXPIR. DATE _____

NAME _____

ADDRESS _____

CITY _____ STATE _____

ZIP _____ PHONE _____

ECOSOFT

Circle Number 8 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED